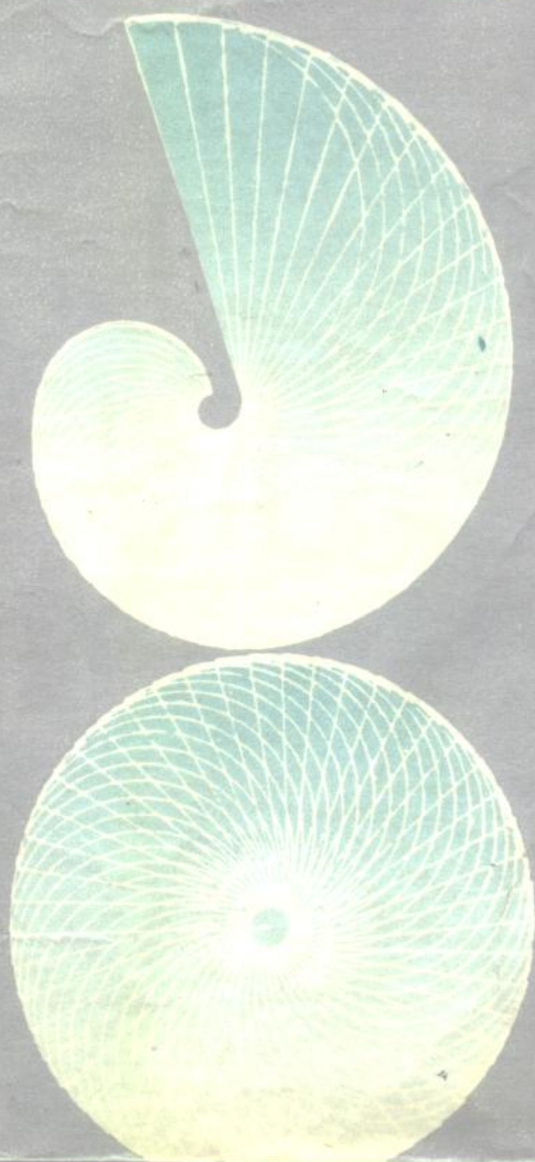
谭浩强 主编

# BASIC 语言

## 结构化程序设计教程



中国科学技术出版社

# BASIC语言

## ——结构化程序设计教程

谭浩强　主编




中 国 科 学 技 术 出 版 社

## 内 容 提 要

本书是在《BASIC 语言》一书（科学普及出版社出版）的基础上重写的一本介绍 BASIC 程序设计的教材，供高等院校非计算机专业人员学习，电视大学、计算机专科学校或培训班皆可用作正式教材，各机关团体和企事业单位的在职工作者，凡具有高中以上文化程度者，均可持以学习计算机知识。本书是按照“程序＝算法＋数据结构＋结构化程序设计方法＋计算机语言”的公式组织教材体系的，因而内容丰富且条理清楚，例题较多且切合应用实际。其特点是突出算法设计，采取结构化程序设计方法编写程序，运用了结构化流程图（N－S 图），体现了科学性、先进性和通俗性的统一，是一本新的学习 BASIC 语言程序设计的好教材。

BASIC 语言
——结构化程序设计教程
谭浩强　主编
责任编辑：朱桂兰
封面设计：赵一东
*
中国科学技术出版社出版（北京海淀区白石桥路 32 号）
新华书店北京发行所发行　各地新华书店经售
国防科工委印刷厂印刷
*
开本：787×1092 毫米 1/16　印张：18.25 字数：440 千字
1990 年 6 月第 1 版　　1990 年 6 月第 1 次印刷
印数：1—20 000 册　　定价：5.00 元
ISBN　7－5046－0160－8/TP・9

# 前　　言

从科学技术发展的角度，可以认为人类已经跨入信息和计算机时代。计算机的应用，已从科学家的实验室步入生产领域、交换领域、管理领域，直至家庭生活领域，从宏观的宇宙航天事业到微观的分子研究，几乎无处不有计算机的存在，并且它的发展正以等比级数的速度向前推进。普及计算机知识，全面推广计算机的应用，不论人们主观意向如何，业已成为历史的必然趋势。

要学习计算机知识和掌握计算机的应用，一般说都要学习和掌握计算机语言。BASIC 语言是一种国内外广泛流行的计算机高级语言，它易于学习，易于掌握，又有实际使用价值，在广大初学者之中得到广泛推广，受到热烈欢迎。近年来我国已有近 1000 万人学习了 BASIC 语言，其中不少人在掌握了 BASIC 语言之后编制了一些能供实际使用的应用程序，解决了一些实际问题。不少人以学习 BASIC 语言为起点不断提高，迈入了计算机应用领域。事实已经说明，BASIC 语言在我国计算机普及和计算机应用中起了重要的作用，并将继续发挥其作用。

1980 年，我和田淑清、谢锡迎同志合编的《BASIC 语言》(科学普及出版社出版) 一书，由于写法通俗易懂，符合初学者特点，受到广大读者的欢迎，9 年内修订三次，重印 30 多次，累计印数已超过 700 万册，创电子和计算机类书籍发行量的世界记录，获得了新闻出版署、中国科协、广播电影电视部等单位颁发的优秀科普图书奖。国内外许多专家对此给予充分肯定和很高的评价。对此，我们表示衷心的感谢。

由于计算机科学技术发展很快，尤其是结构化程序设计方法的提出，有必要重新编写一本 BASIC 程序设计的教材，以满足实际使用中的需要。在近年召开的几次学术会议上不少同志也恳切地表示了这个愿望，希望我能尽快地重写出一本书，以推动计算机语言教学的进一步深入和提高。我在多年教学实践基础上，广泛吸取了国内外同类书籍的有益经验，经过反复斟酌思考，根据国内高等学校非计算专业的实际状况和要求，重新拟订了大纲，终于在 90 年春节之夜，最后完成了本书的全部编写工作。

一般说程序是算法、数据结构、结构化程序设计方法和计算机语言这四个因素结合的产物，可以用“程序＝算法＋数据结构＋结构化程序设计方法＋计算机语言”这一公式来表达。离开计算机语言就谈不上程序设计，要编写程序就必须熟练地掌握一种计算机语言。但是语言只是一种用来描述算法的工具，算法设计是程序的核心。因此本书不把重点放在介绍 BASIC 语言的语法规则上，那样将是枯燥无味而且将成为无源之水，学了语言后仍不会编写程序。本书突出算法设计，使读者能够学到在拿到一个问题以后怎样去构思解决问题的步骤。程序处理的对象是数据，因此要研究数据的属性和数据间的联系，这就是数据结构。采用不同的数据结构会有不同的算法。本书对数据结构也给以必要的重视，把它和算法结合在一起研究。在编程方法上本书全部采用结构化程序的方法，采用 N－S 结构化流程图（书中对 N－S 图作了一定的补充和发展），以帮助读者从开始学习计算机程序设计时就养成一种良好的风格和习

41521

惯。我们认为本书的特点和体系是符合广大读者需要的。本书内容充实，有一定的深度，在学习完本书后能够编写出一般的 BASIC 程序。

我们认为对计算机的初学者和高等学校中非计算机专业学生来说，学习计算机知识的目的在于应用，要在有限的时间内学习到有关的基本概念和初步掌握应用计算机的能力。不能照搬计算机专业教材的内容、体系和方法。我们编写教材的原则是："实用、新颖、清晰、通俗、层次"，即实用性强、内容新颖、概念清晰、通俗易懂、层次配套。我们认为，教材不仅是把具体的知识传播给读者，更重要的是如何组织教材体系，如何使读者易于理解、循序渐进、便于自学。写书和讲课都要研究读者和学生的认识规律，研究心理学，这点对计算机的初学者尤为重要。本书的叙述力求通俗易懂，把难点分解，使台阶变小，相信本书会吸引读者兴趣盎然地进入计算机的天地。

本书的叙述不从规则、定义、概念出发，而是从具体问题入手，提出问题，分析问题，讨论解决问题的方法，最后引出规律和结论。根据我们的经验，这种方法是有效的、事半功倍的，定能受广大读者欢迎。

由于 BASIC 的版本很多，在一本书内不可能也不必要介绍所有的 BASIC 版本。我们选择目前在世界上使用最广泛的 MS - BASIC (Microsoft BASIC，或称 MBASIC) 为语言背景，它能在长城 0520，IBM - PC、浪潮 0520，王安 PC，APPLE Ⅱ（需加 Z80 插板，并在 CP/M 操作系统支持下)、ALTOS 等多种计算机系统上运行，功能比较强。它虽然不是真正的结构化语言，但已具有结构化的语句，可以利用它进行结构化程序设计。我们认为这是符合我国目前情况的。考虑到 BASIC 的创始人 John G Kemeny 和 P. J. Plauger 已经根据美国新的 BASIC 标准提出了 True BASIC（但在我国目前还未普遍使用)，为了使读者在以后使用它不致感到陌生和困难，在本书第十四章对 True BASIC 作了简要介绍。尽管 BASIC 的版本多，不同的计算机系统上使用着不同的 BASIC 版本，但它们是大同小异的，不同系统上所用的 BASIC 的基本部分是相同的。在学习一种 BASIC 之后，很容易举一反三，通过自学或查阅资料很快地掌握其它版本的 BASIC 语言。虽然本书以 MS - BASIC 为语言背景，但它是一本通用的教材。如果读者所用的计算机系统不能使用 MS - BASIC，只需参考有关资料作一些补充或修改即可。由于本书重点不在介绍 BASIC 语法，因此这个问题不会成为学习本书的障碍。

本课程不是一门纯理论性的课程，而是一门实践性很强的课程，学习的目的在于应用，在学习本课程时应当特别注意培养独立的编程序的能力和上机调试程序的能力。一般应保证不少于 20～30 小时的上机练习时间。

在本书的前十二章中没有包括有关高等数学知识的内容，具有高中毕业文化程度的读者都能掌握。在第十三章中介绍了常用算法程序举例，包括了一些比较复杂的算法，其中有些需要有高等数学的知识。读者可以根据不同情况选学第十三章的内容。在学习前面各章时，可以从第十三章选择一些例题结合学习，不一定等到学完第十二章才学习第十三章。

本书的例题较多。但是应当说明，这些程序只是为了帮助读者具体地理解各章中所叙述的内容，从教学的角度介绍的程序例子。这些程序的编写方法不是唯一的，甚至不一定是最好的。为了节约篇幅，程序中使用注释语句不多。读者在学完本书后，完全有可能对本书的程序作一定的补充、修改和完善。

为了帮助读者学习，拟编写出版《BASIC 程序设计习题集与上机指导》，内容包括本书各章重点和难点讲解、补充例题和本书各章习题参考解答以及在几种常用的计算机系统上如何

运行 BASIC 程序和上机实验指导等，由中国科学技术出版社出版。

应当说明，学习 BASIC 程序设计只是学习计算机知识的入门。不要认为学了 BASIC 语言会编写一些简单的程序就等于“学会了计算机”了。计算机科学技术浩如瀚海，包含许多分支学科，内容十分丰富。学习 BASIC 只是入门的第一步。即使是程序设计语言，世界上常用的也有几十种。每一种计算机语言都有自己的应用领域。它们是互相补充而不是互相排斥的。在掌握 BASIC 语言之后，在需要时再进一步学习其它计算机语言是不会太困难的。

本书只是初步介绍程序设计的知识，在学习本书之后应当进一步提高，尤其是需要通过实践深入学习和掌握结构化程序设计方法、技巧和风格。我们建议在学完本课程后，安排一次大作业．完成具有一定规模（约 200 行左右）的程序设计。为了帮助读者进一步学习结构化程序设计知识，拟编写一本《实用程序设计》，作为本书的参考书或后续用书，仍由中国科学技术出版社出版。

参加本书编写的还有上海工业大学卜家岐老师和清华大学谭恩慈老师。卜家岐老师编写了本书的第九、十一、十四章和第十二章的最后二节。谭恩慈老师编写第七、八章。谭浩强编写了一、二、三、四、五、六、十、十二（前三节）、十三章，并负责全书的组织、修改、统稿和审定。徐燕、薛淑斌同志参加了部分程序的调试工作。中国计算机学会和全国高等学校计算机基础教育研究会对 BASIC 语言的推广以及本书的编写工作给予了很大的支持，许多专家和高校的教师多年来对我们的工作给予很多的关心和帮助，广大读者给了我们极大的鼓励。中国科学技术出版朱桂兰副编审以极大的热忱有效地组织了本书的出版工作，使本书得以在很短的时间内出版，在此，一并表示由衷的感谢。

谭浩强

1990. 1. 29.

## 作 者 介 绍



谭浩强教授，北京自动化工程学院副院长，全国高等院校计算机基础教育研究会副理事长，中国共产党党员。1934 年 11 月生，广东省台山县人。1952 年毕业于上海市上海中学。中学时代就参加了团委的领导工作。中学毕业后被派到复旦中学任专职政治辅导员，同期加入中国共产党。1953 年秋考入清华大学电机系，后转自动控制系，1958 年毕业。大学毕业后，留校工作。1956 年至 1959 年被选为清华大学学生会主席。并担任北京市学联副主席和全国学联执行委员会委员。1957 年 2 月，他作为全国学联代表之一，曾受到毛泽东、周恩来、朱德和刘少奇等党和国家领导人的接见。1958 年被选为第三届北京市人民代表大会代表。1958 年 8 月曾作为中国学生代表团成员之一，参加了第五届国际学联代表大会。

1960 年 8 月，因工作成绩突出，曾被评为"北京市文教战线先进分子"，出席了"北京市文教战线群英会"。作为一名优秀的共青团工作干部，由他执笔撰写的一些有关学生工作经验的总结材料曾受到有关领导部门的好评，并在全国性政治工作会议上被印发介绍。

长期从事青年学生工作，使他对学生的思想感情、兴趣爱好、理想追求以及实际需要都有深刻的观察与理解，因而能在青年学生中间卓有成效地进行工作并取得优异的成绩，是当时青年学生团体中具有一定影响的人物。这种影响却使他在"十年浩劫"中受到不公正的待遇，受批判审查、下放劳动共达 6 年之久。

粉碎"四人帮"后，作者在清华大学除继续担任一定的党政工作外，1979 年还开始兼事计算机教学和普及教育。80 年代初是中国计算机应用的起步时代。在这一时代到来之际，作者能以其敏锐的预感、惊人的毅力和潜心治学的精神，在担任繁重的工作的情况下，及时投入计算机的基础教育工作并很快成为这方面的专家，实为难能可贵。80 年代初他就较早地在清华大学为上千名教师和研究生讲授了计算机语言课，并在中央电视台和中央电视大学系统讲授了 BASIC、FORTRAN、COBOL 和 PASCAL 四种计算机语言，收学人数超过 100 万人。作者的这些教学活动，对推动我国计算机语言的普及教育，可说是作出了重要的并具有开创性的贡献。

1980 年以来，在从事普及计算机语言教育的同时，作者本人或与他人合作编写了 40 本有关计算机程序设计语言教材或参考书，共约 1200 万字，累计出版、发行超过 1100 万册。其中，作者与田淑清、谢锡迎合编并由科学普及出版社出版的《BASIC 语言》一书，自 1980 年出版以来，已再版、重印 30 多次，总发行量达 700 万册，创电子和计算机类图书发行量的世界记录，受到中国科协、新闻出版署和广播电影电视部等领导部门的表彰和奖励，许多报纸、广播电台和电视台对此都作过专门报道。《FORTRAN 语言》（谭浩强、田淑清编著）和《COBOL 语言》（谭浩强编著）也分别出版、发行 110 万和 70 万册，发行量都居全国同类图书的前列。这些有关计算机程序设计语言教材，受到广大读者的好评。

1986 年 11 月，作者应邀出席了在东京召开的国际地区性计算机教育会议，在会议上作者宣读了他所撰写的题为《高等学校中计算机基础教育》的论文，受到与会者的重视。1986 年，获北京市高等学校教学成果一等奖，1989 年他被北京市人民政府授予"有突出贡献专家"的称号。1990 年，经中国科普作家协会第三次代表大会审定，表彰他为"有突出成绩的科普作家"。

1985 年 1 月，谭浩强教授调离清华大学，接任现职——北京自动化工程学院副院长。他长期坚持"双肩挑"的方向，在担负繁重的学校行政工作的同时，仍坚持从事计算机的教学工作，上述许多教学活动和编著教材的工作就是在担任自动化学院领导工作期间进行的。本书便是在这种情况下作者推出的新作。

愿借本书出版之机，将谭浩强教授简要经历向读者作一介绍。

中国科学技术出版社<br>1990 年 5 月

# 目　录

第一章　关于计算机的一般知识 …… (1)
§1.1　电子计算机的特点和用途 …… (1)
§1.2　电子计算机的基本结构 …… (2)
§1.3　数据在计算机内的存贮方式 …… (4)
1.3.1　内存的组织形式 …… (4)
1.3.2　数据的二进制表示法 …… (5)
1.3.3　八进制和十六进制 …… (7)
§1.4　机器语言与高级语言 …… (8)
1.4.1　机器语言 …… (8)
1.4.2　高级语言 …… (9)
§1.5　计算机的硬件和软件 …… (10)
§1.6　利用计算机解决实际问题的步骤 …… (11)
习题 …… (12)
第二章　最简单的 BASIC 程序分析和流程图 …… (13)
§2.1　BASIC 语言的基本特点 …… (13)
§2.2　BASIC 程序的构成和基本规则 …… (14)
§2.3　变量 …… (16)
§2.4　常量 …… (16)
§2.5　标准函数 …… (17)
§2.6　表达式 …… (18)
2.6.1　算术运算符 …… (18)
2.6.2　算术表达式 …… (19)
2.6.3　运算规则 …… (19)
§2.7　用传统流程图表示算法 …… (20)
§2.8　结构化程序设计要点和 N—S 结构化流程图 …… (22)
习题 …… (26)
第三章　顺序结构程序设计 …… (28)
§3.1　赋值语句(LET 语句) …… (28)
§3.2　输出语句(PRINT 语句和 LPRINT 语句) …… (29)
3.2.1　PRINT 语句的作用 …… (29)
3.2.2　PRINT 语句的输出格式 …… (31)
3.2.3　实数的输出形式 …… (33)
3.2.4　LPRINT 语句 …… (34)
§3.3　键盘输入语句(INPUT 语句) …… (34)
§3.4　读数语句(READ 语句)和置数语句(DATA 语句) …… (37)
§3.5　恢复数据区语句(RESTORE 语句) …… (38)
§3.6　三种提供数据的语句比较 …… (40)
§3.7　END 语句和 STOP 语句 …… (41)

3.7.1 END 语句 …… ( 41 )
3.7.2 STOP 语句和 CONT 命令 …… ( 41 )
§3.8 程序举例 …… ( 42 )
§3.9 怎样输入和运行一个 BASIC 程序 …… ( 45 )
习题 …… ( 46 )
第四章 选择结构程序设计 …… ( 49 )
§4.1 概述 …… ( 49 )
§4.2 GOTO 语句 …… ( 49 )
§4.3 IF 语句的概念 …… ( 49 )
§4.4 关系表达式和逻辑表达式 …… ( 51 )
4.4.1 关系表达式和关系运算符 …… ( 51 )
4.4.2 逻辑表达式和逻辑运算符 …… ( 53 )
§4.5 IF 语句的嵌套 …… ( 55 )
§4.6 多分支选择语句(ON－GOTO 语句) …… ( 59 )
§4.7 程序举例 …… ( 63 )
习题 …… ( 67 )
第五章 循环结构程序设计 …… ( 70 )
§5.1 概述 …… ( 70 )
§5.2 用 IF 语句和 GOTO 语句实现循环 …… ( 71 )
§5.3 用 WHILE 语句实现循环 …… ( 74 )
§5.4 用 FOR－NEXT 语句实现循环 …… ( 77 )
5.4.1 循环语句的结构 …… ( 78 )
5.4.2 循环语句的执行过程 …… ( 79 )
5.4.3 执行 FOR 循环中的一些问题 …… ( 79 )
§5.5 循环语句应用举例 …… ( 83 )
§5.6 循环的嵌套 …… ( 86 )
习题 …… ( 89 )
第六章 数组 …… ( 92 )
§6.1 数组和数组元素的概念 …… ( 92 )
§6.2 一维数组 …… ( 93 )
§6.3 二维数组 …… ( 96 )
§6.4 数组说明语句(DIM 语句) …… ( 98 )
§6.5 程序举例 …… ( 100 )
习题 …… ( 107 )
第七章 函数与子程序 …… ( 110 )
§7.1 BASIC 标准函数的分类及应用 …… ( 110 )
§7.2 自定义函数 …… ( 115 )
§7.3 子程序 …… ( 118 )
7.3.1 子程序的概念 …… ( 118 )
7.3.2 子程序的结构和子程序的调用 …… ( 119 )
7.3.3 子程序的嵌套 …… ( 122 )
7.3.4 IF－GOSUB 语句和 ON－GOSUB 语句 …… ( 124 )
7.3.5 子程序的作用 …… ( 125 )

习题 …… ( 125 )
第八章 字符处理 …… ( 127 )
§8.1 概述 …… ( 127 )
§8.2 字符串变量 …… ( 127 )
§8.3 字符串的输入 …… ( 128 )
8.3.1 用 READ/DATA 语句向字符串变量赋值 …… ( 128 )
8.3.2 用 INPUT 语句给字符串变量赋值 …… ( 129 )
§8.4 字符串表达式 …… ( 130 )
§8.5 字符串的比较 …… ( 131 )
§8.6 字符串数组 …… ( 134 )
§8.7 子字符串 …… ( 136 )
§8.8 有关字符串运算的函数 …… ( 138 )
习题 …… ( 141 )
第九章 屏幕控制和作图 …… ( 144 )
§9.1 屏幕控制语句 …… ( 144 )
9.1.1 LOCATE 语句和 WIDTH 语句 …… ( 144 )
9.1.2 CSRLIN 和 POS 函数 …… ( 146 )
9.1.3 KEY OFF 和 KEY ON 语句 …… ( 146 )
9.1.4 CLS 语句 …… ( 146 )
9.1.5 SCREEN 语句 …… ( 148 )
9.1.6 字符显示模式下的 COLOR 语句 …… ( 149 )
9.1.7 图形显示模式下的 COLOR 语句 …… ( 152 )
§9.2 画点和画线 …… ( 153 )
9.2.1 画点语句 …… ( 153 )
9.2.2 画线语句 …… ( 154 )
9.2.3 DRAW 语句 …… ( 157 )
§9.3 画圆、椭圆和圆弧 …… ( 158 )
§9.4 图形着色 …… ( 160 )
习题 …… ( 162 )
第十章 输入输出设计 …… ( 164 )
§10.1 输入输出技术 …… ( 164 )
§10.2 格式输出 …… ( 166 )
10.2.1 用 PRINT USING 语句输出数值 …… ( 166 )
10.2.2 用 PRINT USING 语句输出字符串 …… ( 169 )
§10.3 汉字的输入输出 …… ( 170 )
§10.4 "菜单"技术 …… ( 174 )
习题 …… ( 176 )
第十一章 文件 …… ( 178 )
§11.1 文件的基本概念 …… ( 178 )
11.1.1 文件的分类 …… ( 178 )
11.1.2 文件与记录 …… ( 178 )
11.1.3 文件名 …… ( 179 )
11.1.4 文件号 …… ( 179 )
§11.2 源程序文件 …… ( 179 )

11.2.1 SAVE 命令 …… (180)
11.2.2 LOAD 命令 …… (180)
11.2.3 KILL 命令 …… (181)
11.2.4 NAME 命令 …… (181)
11.2.5 FILES 命令 …… (181)
11.2.6 MERGE 命令 …… (181)
11.2.7 RUN 命令 …… (182)
§11.3 顺序文件 …… (182)
11.3.1 顺序文件的打开和关闭 …… (183)
11.3.2 顺序文件的输出(写顺序文件) …… (184)
11.3.3 顺序文件的输入(读顺序文件) …… (185)
11.3.4 顺序文件的修改 …… (187)
§11.4 随机文件 …… (189)
11.4.1 随机文件的打开和关闭 …… (189)
11.4.2 随机文件的缓冲区和 FIELD 语句 …… (190)
11.4.3 随机文件的输出(写随机文件) …… (191)
11.4.4 随机文件的输入(读随机文件) …… (193)
习题 …… (198)
第十二章 结构化程序设计方法和编程技术 …… (199)
§12.1 软件工程方法与结构化程序设计 …… (199)
§12.2 选择数据类型 …… (202)
§12.3 程序风格 …… (205)
§12.4 链接与覆盖 …… (207)
12.4.1 全覆盖链接语句 …… (207)
12.4.2 全不覆盖(合并)链接语句 …… (209)
12.4.3 部分覆盖链接语句 …… (209)
§12.5 陷阱技术 …… (211)
12.5.1 出错陷阱 …… (212)
12.5.2 功能键陷阱 …… (214)
12.5.3 时钟陷阱 …… (214)
习题 …… (215)
第十三章 常用算法程序举例 …… (217)
§13.1 穷举法 …… (217)
§13.2 递推法 …… (219)
§13.3 求函数的定积分 …… (220)
13.3.1 矩形法 …… (221)
13.3.2 梯形法 …… (222)
13.3.3 辛普生法 …… (224)
§13.4 求一元方程的近似根 …… (226)
13.4.1 迭代法 …… (227)
13.4.2 牛顿迭代法 …… (227)
13.4.3 弦截法 …… (229)
13.4.4 二分法 …… (231)
§13.5 排序方法 …… (232)
13.5.1 起泡法排序 …… (232)

13.5.2 插入法排序 …… (234)
13.5.3 希尔法排序 …… (235)
§13.6 矩阵运算 …… (237)
§13.7 计算机模拟 …… (239)
13.7.1 确定性模拟 …… (239)
13.7.2 机率性模拟 …… (243)
§13.8 用高斯消元法解一元联立方程组 …… (246)
习题 …… (248)
第十四章 True BASIC 简介 …… (251)
§14.1 引言 …… (251)
§14.2 True BASIC 的结构化语句 …… (251)
14.2.1 True BASIC 的选择结构 …… (252)
14.2.2 True BASIC 的循环结构 …… (255)
§14.3 True BASIC 程序的模块化 …… (258)
14.3.1 函数的定义和调用 …… (258)
14.3.2 子程序的定义和调用 …… (259)
14.3.3 库文件 …… (259)
§14.4 True BASIC 的绘图 …… (260)
14.4.1 图形窗口坐标 …… (260)
14.4.2 作图 …… (261)
14.4.3 着色 …… (262)
14.4.4 图形中的正文设置 …… (263)
14.4.5 动画 …… (264)
14.4.6 图画功能 …… (265)
§14.5 True BASIC 中的矩阵运算语句(MAT 语句) …… (267)
§14.6 怎样使用 True BASIC …… (268)
14.6.1 进入 True BASCIC …… (268)
14.6.2 屏幕上的窗口 …… (269)
14.6.3 退出 True BASIC …… (269)
14.6.4 编辑命令 …… (269)
14.6.5 True BASIC 的文件处理和程序控制命令 …… (270)
习题 …… (271)
附录Ⅰ 常用字符与 ASCII 代码对照表 …… (272)
附录Ⅱ MS-BASIC 语句和函数一览表 …… (273)
参考文献 …… (277)

# 第一章　关于计算机的一般知识

## §1.1　电子计算机的特点和用途

电子计算机的出现是人类生产发展的必然产物，是现代科学技术的重要标志。自 1946 年产生世界上第一台电子计算机 ENIAC 以来，计算机的生产、研究和应用以迅猛的速度发展着。现在，电子计算机已渗入到人类生产和生活中的几乎一切领域，可以说，没有电子计算机就谈不上现代化。电子计算机的知识应当成为当代知识分子的知识结构中的不可缺少的重要组成部分。

电子计算机具有以下特点。

(1)运算速度快。巨型机的运算速度已达每秒几十亿次，是传统计算工具(如算盘、计算尺、手摇计算机等)所不能比拟的。

(2)精确度高。计算机能提供十几位以上的有效数字。

(3)具有“记忆”和逻辑判断能力。它能把计算步骤、原始数据、运算结果存贮在计算机内，这是电子计算机与其它计算装置的一个重要区别。计算机还能进行逻辑判断，并根据判断结果自动决定以后执行的命令。

(4)计算机内部的操作运算，都是在程序的控制下自动完成的，人可不必进行干预。

概括地说，电子计算机是一个以高速进行操作、具有内部存贮能力、由程序控制操作过程的自动电子装置。

电子计算机的用途十分广泛，据估计，应用计算机的领域已超过 5000 个，概括起来，可以分为以下几大类。

(1)数值计算，或称科学计算。例如工程设计、天气预报、地震预测等。1948 年，美国原子能研究中有一项计划，要做 900 万道运算，需要由 1500 名工程师计算一年。当时利用了一台初期的计算机，只用了 150 小时就完成了。有人估计，美国现有电子计算机所完成的工作量，如果用人工，需要 4000 亿个人才能够完成。

早在 1671 年，德国数学家莱布尼兹说过：“让一些杰出的人才象奴隶般地把时间浪费在计算上是不值得的。”他渴望有朝一日能有计算机把科学家从这种奴隶般的计算中解放出来，这个愿望现在实现了。

(2)数据处理，或称信息处理。计算机不仅能高速地进行数值计算，而且能对大量的数据进行有效的加工和处理(如分类、排序、变换、检索、制表等)。例如，工农业计划的制订、人口统计处理、企业经济管理、银行业务、预订机票、档案管理、图书检索、学籍管理、编辑排版、卫星图象分析等等。

(3)过程控制，或称实时控制。计算机能及时采集检测数据，按最优方案实现自动控制。例

如炼钢过程的计算机控制、高射炮自动瞄准系统、飞行控制调度等。计算机用于生产过程自动化，大大提高了生产效率和产品质量，大大节约了劳动力。

(4)计算机辅助设计，简称 CAD (Computer Aided Design)。用计算机辅助人们进行设计工作，如设计飞机、房屋、服装、集成电路等，使设计工作自动化或半自动化。近年来还发展了"计算机辅助制造"(简称 CAM，Computer Aided Manufacture)，实现无图纸加工。

计算机辅助教学(简称 CAI，Computer Aided Institute)，是利用计算机来辅助进行教学，把教学内容编成"课件"，学生可以根据自己的程度选择不同内容，可使教学内容多样化、形象化，便于因才施教。我国已开展 CAI 的研究和开发。

(5)人工智能。这是计算机应用的新领域。主要研究如何用计算机来"模仿"人的智能，也就是使计算机具有"推理"和"学习"的功能。例如，计算机辅助诊断就是模拟医生看病，计算机可以开药方写假条；计算机还可以下棋、作曲、翻译，机器人和机械手可以完成人们难以完成的操作。人工智能应用的前景十分广阔。

计算机问世初期，主要应用于数值计算，"计算机"也因而得名。现在，计算机在非数值运算方面的应用远远超过在数值运算方面的应用。其实，计算机的名字称为"信息处理机"更为确切。也有人称之为"电脑"，意为人脑的"延长"。

## §1.2 电子计算机的基本结构

计算机算题的过程和人利用算盘算题差不多。只要知道算盘是怎样算题的，就可以懂得计算机的算题过程和它的基本结构。

**(一)利用算盘算题的步骤和需要的设备**

如果我们要计算 86－25×3＝？要经过几个步骤。

①根据给定的题目想好计算方法和计算步骤，并把计算公式、计算步骤、原始数据等写在纸上。在本例中：

计算公式是：A－B×C＝D；

计算步骤是：先算 B×C，再算 A－B×C；

原始数据是：A＝86，B＝25，C＝3。

②在算盘上进行计算，规则是先乘除，后加减。先算 25×3＝75，我们把这中间结果 75 写在纸上以备调用。然后在算盘上拨上 86，再做减法，86－75＝11。

③把最后结果 11 记录在纸上。

到此全部结束。

从上面可以看出：要完成这一道题目，必须具有：

1)能进行运算的装置，即算盘。

2)能存放题目、计算步骤、原始数据、中间结果和最后结果的装置，即纸张。在整个计算过程中，把需要记录的数据都"记存"在纸上，需要"取出"时再按纸上的数值拨到算盘上。

3)进行控制的装置。上述计算都是在人脑的操纵下进行的，由手去执行。

**(二)计算机解题所需的设备**

电子计算机的计算过程与算盘相仿，只是它由机器代替人。因此，和用算盘算题一样，必须具备几种设备。

①运算器。进行运算，相当于算盘。

②计算机必须能保存和记录原始数据、运算步骤、以及中间结果。也就是说需要“记忆装置”，即存贮器。它相当于纸和笔。

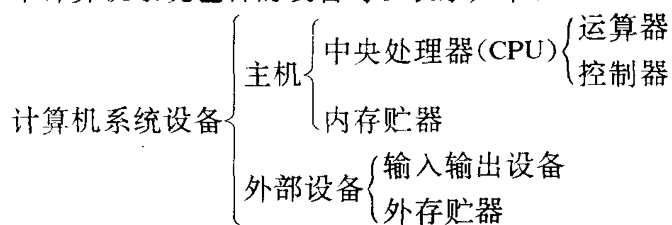
③计算机要有一个代替人的脑和手的作用、支配机器进行自动控制的**控制器**。它是计算机的“神经中枢”。由它统一指挥和控制计算机各部分的联系。例如上例中从纸上“取”一个数据到算盘上，和把结果“存”到纸上，是由人脑和手完成的。而在计算机中则全由控制器发出命令：什么时候取数，从什么地方取数，送到什么地方，进行什么运算，算完后的结果送到哪里等等。

④输入和输出设备。如果只有上述三种设备，计算机还不能工作。因为要算题，人们必须事先把原始数据和规定的计算步骤送到计算机中去，而计算的结果又要由计算机输出来。这种人和计算机联系的桥梁，称为输入或输出设备。

常用的输入设备有：显示器键盘、卡片输入机、软盘输入机等，常用的输出设备有：终端显示器（荧光屏）、行式打印机。磁带机和磁盘机既可看作是输入输出设备，也是计算机的“外存贮器”，磁带或磁盘可用来存贮信息（包括程序和数据）。

运算器、控制器合称为“中央处理器”（简称 CPU，Centre Processing Unit），由它对数据进行各种操作，存贮器只是存放信息的，本身不产生任何操作。

一个计算机系统包含的设备可以表示如下：




计算机系统的设备又称为计算机硬件系统。

计算机各部分联系示意图见图 1.1。

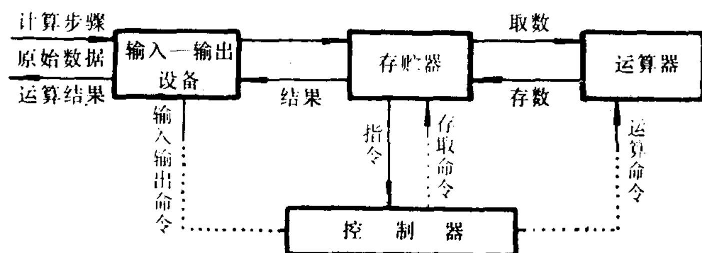



图 1.1

可以看到：在计算机内部存在三种“信息流”。

（1）指令流。图中以粗线表示。从存贮器中将事先已存放在那里的计算机指令逐条送到控制器中，控制器根据这些指令的要求向计算机各部分发出控制命令。

（2）控制流。图中以虚线表示。它包括：①控制器通知从存贮器存取数据的命令；②通知运算器进行运算的命令；③通知输入/输出设备将数据读入存贮器或将内存中的数据通过输出设备输出的命令。

（3）数据流。图中以实线表示。包括：①由输入设备将数据输入到存贮器；②由存贮器将数据送到运算器以便进行运算，运算后再将运算结果送回存贮器；③从存贮器将数据送到输出设

备，由输出设备将数据记录在外部介质（如打印纸）上。

**（三）计算机算题的简单过程**

仍以 86－25×3＝？为例，说明计算机的工作过程。

第一步：由输入设备（如显示器键盘）将事先编好的计算步骤（即程序）和原始数据（86，25 和 3）输入到计算机的存贮器中存放起来。

第二步：输入运行命令，存放在存贮器中的程序（它是由若干条指令组成的）将指令逐条送到控制器中，在控制器的控制下，发出相应的命令按指定的计算步骤自动进行以下操作：

（1）从存贮器中取被乘数 25 和乘数 3 到运算器，进行乘法运算。运算后得乘积 75。

（2）把运算器中的中间结果 75，送回到存贮器存放，以备调用。

（3）从存贮器中取出被减数 86 和减数 75 到运算器，进行相减。在运算器中求得相减的结果 11。

（4）将运算器中的最后结果 11 送回存贮器。

第三步：把存贮器中的最后结果 11 送到输出设备，把这个最后结果显示在荧光屏上或打印在纸上。到此解题过程结束。

## §1.3 数据在计算机内的存贮方式

### 1.3.1 内存的组织形式

电子计算机内的存贮器（称内存贮器，简称内存）是由千千万万个小的电子线路单元组成的，每一个单元有两种稳定的工作状态（例如三极管的截止和导通），它们以 0 和 1 代表。因此电子计算机存贮的信息是用二进制形式存贮的。每一个小元件中存放一个 0 或 1 的信息。关于内存，常用到以下一些术语。

**位**，又称比特（bit）。每一个能代表 0 和 1 的电子线路称为一个二进制位。一个存贮器就是一个包含许许多多个二进位的电子单元的庞大电路。目前的存贮器多采用集成化技术，把几万或几十万个电子元件做成一个集成电路，体积小，耗能少，速度快。

**字节**，又称拜特（byte）。由若干个二进制位组成一个字节，多数计算机以 8 个二进制位组成一个字节。

**字**（word）。由若干个字节组成一个存贮单元，称为“字”。一个存贮单元中存放一条指令或一个数据。如果一个计算机以 32 个二进制的信息表示一条指令，就称这台计算机的“字长”为 32 位。

**地址**。为便于管理，对每个存贮单元编一个存贮单元号，这就是“地址”。通过地址可以找到所需的存贮单元，可以取出其中存贮的信息（或向指定的存贮单元存入信息）。这如同旅馆中有许多房间，给每个房间编一个房号，只有通过房间号才能找到所要找的客人。请不要将存贮单元的“地址”和存贮单元中的“内容”这两个不同的概念混淆。正如同房间号和住在其中的旅客是两回事一样。见图 1. 2。

| | | | | | |
|---|---|---|---|---|---|
| 001 | 002 | 003 | 004 | 005 | 006 |
| 007 | 008<br>25 | 009 | 010 | 011 | 012 |
| 013 | 014 | 015 | 016<br>3 | 017 | 018 |
| 019 | 020 | 021 | 022 | 023 | 024<br>75 |

图 1. 2

假设在地址为 008 的存贮单元中放一个数 25，在地址为 016 的存贮单元中放数 3，分别从这两个存贮单元中将数取到运算器进行相乘，然后将乘积 75 送回地址为 024 的存贮单元。

存贮器、存贮单元、字节、位之间的关系见图 1. 3。

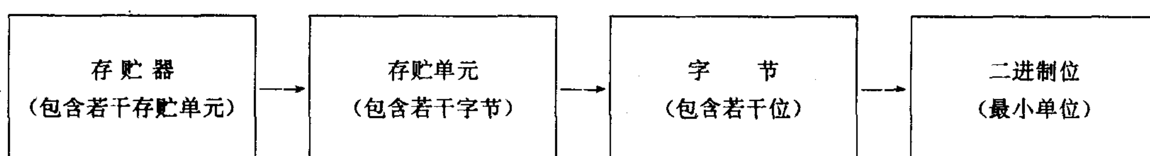



图 1. 3

## 1.3.2 数据的二进制表示法

一个十进制数必须先化成二进制形式才能存放到计算机的存贮单元中。二进制的基本原则是“逢二进一”。人们生活中常遇到的鞋、袜、筷子、手套等就是逢二进一的实例。计算机之所以采用二进制只是因为许多电子元件具有两种稳定工作状态（代表 0 和 1），也就是说容易实现。

表 1. 1 表示 10 以内的十进制数与二进制数的对照。

表 1. 1

| 十进制数 | 化为以 2 为底的指数形式 | 二进制数 |
|---|---|---|
| 0 | $0\times2^0$ | 0 |
| 1 | $1\times2^0$ | 1 |
| 2 | $1\times2^1+0\times2^0$ | 10 |
| 3 | $1\times2^1+1\times2^0$ | 11 |
| 4 | $1\times2^2+0\times2^1+0\times2^0$ | 100 |
| 5 | $1\times2^2+1\times2^1+1\times2^0$ | 101 |
| 6 | $1\times2^2+1\times2^1+0\times2^0$ | 110 |
| 7 | $1\times2^2+1\times2^1+1\times2^0$ | 111 |
| 8 | $1\times2^3+0\times2^2+0\times2^1+0\times2^0$ | 1000 |
| 9 | $1\times2^3+0\times2^2+0\times2^1+1\times2^0$ | 1001 |

一个十进制数 $F$ 可表示为：

$$(F)_{10}=a_0\cdot2^n+a_1\cdot2^{n-1}+\cdots+a_{n-1}\cdot2^1+a_n\cdot2^0$$

则 $a_0$、$a_1$、$a_2$、$a_3$、$\cdots a_n$ 就是 F 在二进制中的表示形式。欲求出 $a_0$、$a_1$、$\cdots a_n$，只需将 $F$ 不断被 2

除，取其余数即可。如求十进制数 11 的二进制形式。

$$\begin{array}{r|l} 2 & 11 \quad (1 \leftarrow a_3 \\ \hline 2 & 5 \quad (1 \leftarrow a_2 \\ \hline 2 & 2 \quad (0 \leftarrow a_1 \\ \hline 2 & 1 \quad (1 \leftarrow a_0 \\ \hline & 0 \end{array}$$

最后一个余数为 $a_0$，从下向上依次为 $a_0, a_1, \cdots a_n$，因此 $(11)_{10} = (1011)_2$。括弧外的注脚“10”和“2”分别表示括弧中的数是十进制数和二进制数。

反之，二进制可化为十进制数，如表 1. 2。

表 1. 2

| 二进制数 | 十进制数 |
|---|---|
| 1 | $2^0=1$ |
| 10 | $2^1=2$ |
| 100 | $2^2=4$ |
| 1000 | $2^3=8$ |
| 10000 | $2^4=16$ |
| 100000 | $2^5=32$ |
| 1000000 | $2^6=64$ |
| 10000000 | $2^7=128$ |
| ⋮ | ⋮ |
| $\underbrace{100\cdots\cdots000}_{n\text{个}0}$ | $2^n$ |

如果一个二进制整数要化为十进制数，只要将它的最后一位乘以 $2^0$，最后第二位乘以 $2^1$，……依此类推，将各项相加就得到用十进制数表示的数。如：

$$\begin{aligned}(101101)_2 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 2^5 + 2^3 + 2^2 + 2^0 \\ &= 32 + 8 + 4 + 1 = (45)_{10}\end{aligned}$$

如果想把一个字符（例如字符“A”或“＄”）存放到计算机内存中，用什么方法呢？显然计算机内是不能直接存贮英文字母或专用字符（如“＄”、“?”、“#”等）的，也要改用二进制形式的代码来存放。即：对每一个字符指定一个二进制代码。目前多数计算机系统采用 ASCII 代码（American standard Code for Information Interchange，即“美国信息交换标准码”），参见本书附录一。例如字符“A”用 01000001 代表，字符“＄”用 00100100 代表。每一个字符用八个二进位（一个字节）存放。如果我们想输入字符“A”，按键盘上的“A”键，系统会自动将“A”转换成 ASCII 代码 01000001，然后送到计算机内存中。

以上简单说明了整数和字符是如何存放到计算机中的。对于带小数的实数，在计算机中是以指数形式存放的，在此不作介绍。

### 1.3.3 八进制和十六进制

由于二进制数位数多，数字冗长，不便记忆和书写，因此常将二进制数分成三位一组，如10110101111 可分为 10，110，101，111 四组，每一组代表一个 0 到 7 之间的数（三位数最大为 111，相当于十进制的 7）。因此，它是八进制数，即“逢八进一”。

$$\underbrace{10}_{2},\ \underbrace{110}_{6},\ \underbrace{101}_{5},\ \underbrace{111}_{7}$$

$(10\ 110\ 101\ 111)_2=(2657)_8$。注意不是十进制的 2657，而是八进制的 2657。它的十进制形式为 1455。

如果将二进制数分成四位一组，就是十六制数（四位的二进位数最大为 1111，相当于十进制的 15，即逢十六进一）。如：

$$\underbrace{101}_{5},\ \underbrace{1010}_{10},\ \underbrace{1111}_{15}$$

在十六进制中把 10 以上的数用 A（代表 10）、B（代表 11）、C（代表 12）、D（代表 13）、E（代表 14）、F（代表 15）表示。因此，$(10110101111)_2=(1455)_{10}=(2657)_8=(5AF)_{16}$。

表 1. 3 举例说明十进制数和二进制数、八进制数、十六制数的对应关系。

表 1. 3

| 十进制数 | 二进制数 | 八进制数 | 十六进制数 |
|---|---|---|---|
| 32 | 110000 | 40 | 20 |
| 62 | 111110 | 76 | 3E |
| 114 | 1110010 | 162 | 72 |
| 241 | 11110001 | 361 | F1 |
| 200 | 11001000 | 310 | C8 |

在应用时，必须弄清楚所接触的数是什么进制的，不要弄错。例如对二进制中的 100 应读作“壹零零”而不应该读为“一百”。

以上只作简单的介绍，以便使读者对二进制数有一初步概念。实际上，在数据输入计算机时，是由计算机系统自动将十进制数转换为二进制数的，不必使用者自己转换，在输出时，由计算机将内存中的二进制数先转换成十进制数，然后输出给用户的。初学者对八进制和十六进制可不必深究，知道有这一回事就行，以便今后查阅资料时不致茫然。

## §1.4 机器语言与高级语言

### 1.4.1 机器语言

要使计算机按人的意图工作，就必须使计算机懂得人的意图，接受人向它发出的命令和信息。人要和机器交换信息就要解决一个“语言”的问题。计算机并不懂人类的语言（无论是中文或英文），例如，我们写 A＋B＝C，机器不能接受。它只能识别 0 和 1 两种状态。长城 0520（与 IBM－PC 兼容）计算机一个字长为 16 位，也就是说，由 16 个二进制数（0 或 1）组成一条指令或其它信息。16 个 0 和 1 可组成各种排列组合，通过线路变成电信号，让计算机执行各种不同的动作。

如：1011011000000000

在某一类型的计算机中，这条指令的作用是让计算机进行一次加法。

又如：1011010100000000

这条指令的作用是让计算机进行一次减法。

以上两条指令只是第 6 和第 7 位不同（从第 0 位，即最左边的一位算起），可以看出，16 个 0 和 1 最多可以组成 $2^{16}$ 个不同的指令或信息。

人要和机器进行联系，就要编出这种由 0 和 1 组成的数字代码。这种计算机能接受的代码，称为**机器指令**。一条指令用来控制计算机进行一个操作内容。它告诉计算机应进行什么运算、哪些数参加运算、这些数存放在什么地方（到哪里去取数）、计算结果应送到什么地方去，等等。

一种计算机的指令的集合称为该计算机的**机器语言**，或者说该计算机的指令系统。正如同用算盘算题一样，每一条珠算口诀就是一条指令，全部口诀之和就是“珠算语言”。也就是说，“语言”是全部指令的总和。为了解某一个问题，可以从该“语言”中选择所需的指令，组成一个指令序列，这称为**机器语言程序**。

用机器语言编写程序是一件十分繁琐的工作，要记住各种代码和它的含义是不容易的。而且编出的程序全是 0 和 1 的数字，直观性差，非常容易出错。程序的检查和调试都比较困难。

不仅如此，不同的计算机系统的线路逻辑是不同的，因此，对不同的计算机，即使是执行同一个操作，其指令也是不同的。或者说，不同的计算机有不同的指令系统（即有不同的机器语言）。一般说，不同型号计算机的机器语言是互不通用的。例如有的计算机字长为 16 位，而有的计算机字长为 8 位，还有的为 32 位，显然，用甲型机器的机器指令编写的程序，拿到乙型机器上是不适用的，需要重新编写程序，显然这是很不方便的。因此说，机器语言是依赖于具体计算机的而不是通用的，它是面向机器的，称为“低级语言”。

由于机器语言与人们习惯用的语言差别太大，难学、难写、难记、难检查、难修改，而且不同机间器又不通用，因此给计算机的推广使用造成了很大的障碍。

### 1.4.2 高级语言

为了解决机器语言（又称为低级语言）的上述缺陷，人们创造了“高级语言”，BASIC 就是高级语言中的一种。

人们进行计算，一般是用数学式子来表达意思，如 Y＝3・sin（A＋B）等，但计算机又不懂得这种“数学语言”。人们设想，能否找到一种过渡性的语言，它比较接近人们习惯的自然语言（如英文）和“数学语言”，又能为机器所接受。譬如，用数学符号写“＋”，计算机就执行加法，写 SIN（X），计算机就计算出 X 的正弦值，写 PRINT A，B，计算机就能打印出变量 A 和 B 的值，如果能做到这点，将为使用者提供极大的方便。

50 年代，创造出一种程序设计语言，它很接近于人们习惯使用的自然语言和数学语言。例如用 BASIC 语言，要想得到 $3\times8\sin(\pi/2)$ 的计算结果，只需写出 PRINT 3＊8＊SIN（3.14159/2）即可，计算机将打印出结果。这种语言，人们容易掌握和理解，容易推广。这种语言不再是面向机器的了，而是面向解题过程的，就是说，不必考虑机器内部构造和不同机器的特点，只要按照解题步骤写出程序，计算机就能执行，这种语言称为“算法语言”。即：用这种语言写程序需要指出计算机一步一步如何做，即要指明算法。这种不依赖于具体计算机的语言被称为“高级语言”。

目前使用较多的算法语言有：

BASIC——一种易学易用又有实际使用价值的计算机语言，最适于初学者使用。

FORTRAN——世界上最早出现的高级语言，适于数值计算。

COBOL——适用于商业和管理领域。

PASCAL——最早出现的结构化语言，适于教学中使用。

PL/1——一种大型的语言，功能强，兼有数值计算和数据处理的功能。

ADA——一种工程化的大型语言，功能很强。

C——近年来得到广泛推广的适于编写系统软件的结构化的语言。

国内外使用的高级语言达几百种之多，每一种语言都有自己规定的语法规则，例如有的高级语言用“PRINT”代表“打印输出”，而有的语言则以“WRITE”代表之。要使用某一种语言必须熟悉该语言的语法规则。

事实上，计算机并不能直接接受和执行用高级语言写的程序（它只能接受 0 和 1 组成的信息），因此必须要有“翻译”，把人们用高级语言写的程序（称为“源程序”）翻译成机器指令的程序，然后再让计算机执行机器指令。这种“翻译”，通常有两种做法，即编译方式和解释方式。编译方式是：事先编好一个称为**编译程序**的机器指令程序，并放在计算机中。把用高级语言写的源程序输入计算机，编译程序便把源程序整个地翻译成用机器指令表示的目的程序。然后执行该目的程序，得到计算结果。如图 1.3（a）所示。

解释方式是：事先编好一个称为**解释程序**的机器指令程序，并放在计算机中。当高级语言源程序输入计算机后，它并不是象编译方式那样把源程序整个地翻译成目的程序，然后再执行该目的程序。而是逐句地翻译，译出一句立即执行，即边解释边执行。见图 1.3（b）。这种方式比编译方式多费机器时间，但可少占计算机的内存。

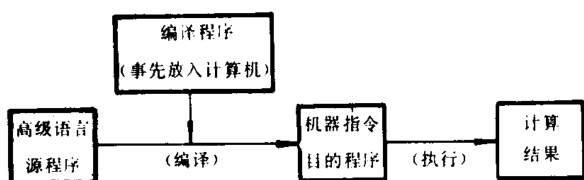



图 1. 3 (a)

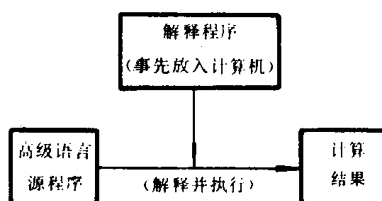



图 1. 3 (b)

FORTRAN、PASCAL、COBOL 等高级语言采用编译执行方式，而大多数 BASIC 高级语言采用解释执行方式（近年已有编译 BASIC）。

由于编译（或解释）程序代替了人工把用高级语言写的源程序翻译为机器指令程序，这就大大节省了使用者的工作量。自从有了高级语言后，一般的科技人员和大、中学生以及职工，都能很快地学会使用计算机，而可以完全不顾什么机器指令，也可以不必深入懂得计算机的内部结构和工作原理，就能方便地使用计算机进行各种科学计算或事务管理等。因此有人说，高级语言的出现是计算机发展中“最惊人的成就”。

使用高级语言写程序还有一个很大的优点，就是它可以适用于不同的计算机，或者说，对不同的计算机具有通用性。用某一种高级语言编写的源程序几乎可不加修改（有时需作一些微小的修改）就能使用在不同的计算机上。这就给使用者带来很大的方便。应当指出：即使是同一种高级语言，对不同型号的计算机来说，它的具体编译系统是不同的。正如把同一篇中文翻译为英文和翻译为法文需要不同的翻译一样。但是这个问题用户不必过虑，在计算机出厂时，已经将该机器所使用的各种语言的编译程序（或解释程序）记录在磁盘或磁带上作为计算机系统的软件同时提供给用户了。

近年来，开始发展一种更为高级的高级语言（或称“超高级语言”），只需指出让计算机“做什么”，而不必指出“怎样做”，计算机全自动完成所需的步骤，显然这又大大前进一步。这就是“非过程化的语言”。例如 DBASE 就属于这类语言。

现在所用的大多数语言仍然是过程化的语言，学习用过程化的语言（算法语言）编写程序，是目前学习计算机技术的一项基本训练。

## §1.5 计算机的硬件和软件

一个计算机系统包括硬件系统和软件系统两大部分。运行一个程序，既需要有必须的计算机设备，还需要有必须的软件环境的支持。

（一）硬件系统，即机器系统，指计算机主机及其外围设备，它包括：控制器、运算器、内存贮器、输入输出设备、外存贮器等。

（二）软件系统，或称程序系统。广义地说，软件泛指程序、运行时所需数据以及程序有关的文档资料。软件系统着重研究如何管理机器和使用机器的问题。也就是研究怎样通过软件的作用更好地发挥计算机的功能。一个不包含任何软件的计算机称为“裸机”。在裸机上只能运行机器语言源程序，显然它的功能是有限的，机器的效能没有得到有效的发挥。

计算机软件主要有两大类：系统软件和应用软件。系统软件是计算机厂商在出厂时提供的，常见的有：汇编程序、编译程序、操作系统、故障诊断程序、控制程序、数据库管理系

统等，用户可以使用它但不应随意修改它。应用程序指计算机用户利用计算机的软硬件资源为某一专门应用目的而开发的程序，例如工资管理程序、桥梁应力计算程序、统计程序等。此外还有一种称之为“工具软件”，即提供一种软件工具以方便用户进行软件开发，例如编辑软件（如 EDLIN，WORDSTAR）、绘图工具（如 AUTOCAD）、电子表格程序等。

要使计算机充分发挥其效能，除了要有较好的硬件外，还要有丰富多样的软件。硬件系统和软件系统构成一个完整的计算机系统。

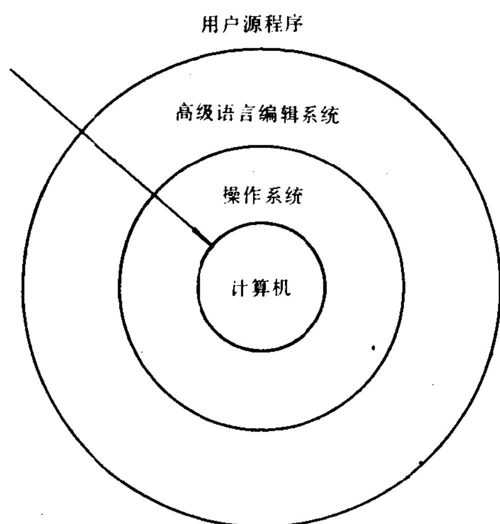


图 1. 4

在软件中最重要的是“操作系统”，它是所有软件的核心，现在几乎所有的计算机都配置了操作系统(Operating Systen，简称 OS)。操作系统是一个庞大的程序，它控制所有在该计算机上运行的程序并管理这个计算机的所有资源。它的指导思想是：充分利用计算机的全部资源，最大限度地发挥计算机系统各部分的作用。通俗地说，它如同大乐队的指挥，使各部分协调有效地工作。BASIC解释程序就是在操作系统管理下进行工作的，或者说，BASIC解释程序只有在操作系统支持下才能工作。图 1.4 表示计算机——操作系统——高级语言编译程序(或解释程序)——用户源程序之间的关系。

它表示：用户编写的源程序是先经过编译系统处理，而编译系统又是在操作系统控制下工作的，只有通过操作系统，它们才能使计算机产生相应的动作。

使用计算机解题的人，除了需要用高级语言编程序外，还应当会用操作系统提供的一些操作命令（如复制一个文件；列出磁盘文件目录、删去一个文件…等）。

许多计算机系统采用分时操作系统，即允许许多用户（几个到几十个）同时使用同一个计算机，以提高计算机使用效率。

## §1.6 利用计算机解决实际问题的步骤

如果求解的是一个比较简单的问题，一般要经过以下几个步骤。

(1) 分析问题，明确问题要求。

(2) 建立数学模型。将一个实际问题（如电路电流计算、导弹轨迹计算…）用数学语言表示（如列出数学式）。

(3) 确定数据结构和算法。解题要考虑数据的组织形式，这就是数据结构。算法是指解题的具体步骤，即考虑好如何一步步地进行操作。一般用流程图来表示算法。

(4) 根据已确定的算法用高级语言编写程序，这一步又称编码 (Coding)。

(5) 调试程序。将写好的源程序上机试运行，输入所需的数据（数据应包括不同的“试验数据”）测试在各种不同情况下程序能否正常运行。如发现程序有错，应修改程序。

(6) 通过调试得到可供正式运行的程序后，上机正式运行，送入正式的数据，得到必要

的运算结果。分析结果是否正确。

(7) 整理资料，写出程序文档（程序说明书），包括：题目；任务要求；原始数据；数据结构；算法（可用流程图表示）；程序清单；运行结果；所用计算机系统配置；操作说明。文档是软件的重要部分，供日后查阅。应养成写文档的良好习惯。

如果要处理的是一个复杂庞大的任务，则应按照软件工程的要求，进行系统分析、系统设计和程序设计。即应经历以下几个阶段。

(1) 问题定义与需求分析。

(2) 总体设计，或称概要设计。将一个大任务划分为若干个子任务，即划分模块。

(3) 详细设计。根据给定子任务（模块）的要求，设计算法和数据结构。

(4) 编写程序和单元测试。

(5) 综合测试与确认运行。

(6) 系统维护。

以上六个阶段称为“软件生存周期”。其中的 (1) 称为“软件定义时期”；(2) ～ (5) 称为“软件开发时期”；(6) 称为“软件维护时期”。一般所说的“程序设计”大体包括以上 (3) (4) 两个阶段。一个大任务不是一个人能完成的，是由系统分析师、系统设计师、高级程序员、程序员共同完成的。

因此，学习 BASIC 语言（或其它高级语言）并用它编写程序，只是学习计算机应用的初步，计算机知识浩如瀚海，入门后还要不断扩展自已的知识，以适应面临的复杂任务。

## 习　　题

1.1　请用你自己的语言说明什么是电子计算机。电子计算机与袖珍电子计算器的主要区别在哪里？

1.2　请你从自己的见闻中举出计算机应用的五个例子。

1.3　请说明一个计算机系统应包括哪些部分(分别说明硬件系统和软件系统所包括的部分)。

1.4　计算机的内存贮器的作用是什么？它本身有无运算功能？

1.5　计算机内存贮器的组织形式是怎样的？什么叫“位”、“字节”、“字”？存贮单元的“地址”是指什么？“地址”与存贮单元中的信息有什么关系？

1.6　计算机内部为什么要采用二进制形式存贮数据和进行运算？为什么不用十进制？

1.7　把下列各十进制数用二进制形式表示：

(1) 92　(2) 128　(3) 136　(4) 246　(5) 1024

1.8　把下列各二进制数化成十进制数：

(1) 1110　(2) 1010　(3) 101111　(4) 11000010　(5) 1011010

1.9　把下面二进制数用八进制和十六进制形式表示：

(1) 10110111　(2) 110001　(3) 0101010101　(4) 1111111111　(5) 1000001

1.10　试述机器语言与高级语言的特点，为什么要使用高级语言？计算机为什么能执行高级语言程序？

1.11　用计算机解题一般应经历哪几个阶段？为什么要书写程序文档？

# 第二章　最简单的BASIC程序分析和流程图

## §2.1　BASIC语言的基本特点

BASIC语言是目前国际通用的计算机算法语言。小型和微型计算机一般都配有BASIC语言，它是一种适合于初学者使用而又实用的计算机高级语言。学会了BASIC，然后再学其它语言（例如F ORTRAN，ALGOL，COBOL，PASCAL，C，PL/1……）也就不难了。

BASIC是Beginner's All-purpose Symbolic Instruction Code（初学者通用符号指令代码）一词的缩写。它是专门为初学者设计的一种计算机语言。

BASIC语言有以下几个特点：

(1) BASIC语言好懂易学。BASIC的常用语句只有二三十种。BASIC的命令和语句中使用的词以及运算符号与英语中使用的词以及数学中的符号差不多，因此比较直观，易于理解和记忆。

例如英语中"PRINT"的意思是"印刷"，在BASIC语言中它也代表"打印"，它使计算机执行一个"打印"的操作，打印出你所需的内容。又如："IF A＞0 THEN PRINT A"这一语句的含义是："若A＞0，则打印A的值"，意思清晰易懂。

(2) 大多数BASIC采取解释执行方式，是会话式的语言，便于人机对话。在输入源程序并运行后，计算机系统会检查语法有无错误，如有错屏幕上会显示出错误信息，用户可以立即在键盘上修改出错的语句，在计算机的终端上边计算边修改，直至得到满意的结果为止。BASIC程序的调试是十分方便的。对于编程序还不大熟练的初学者，程序中难免有错，用BASIC修改错误是很方便的。

(3) BASIC除了可以用"程序方式"通过运行程序得到运算结果外，还提供一种"命令方式"，即允许在终端的键盘上直接进行运算（称键盘运算）和执行某些命令，而不必专门编写一段程序。这时如同使用计算器一样。如在键盘上打入：PRINT (3+SIN (0)/4，则计算机立即打印出结果0.75来。它又称为"直接方式"。这种功能在检查和调试程序中是很有用的，为了检查程序的某一部分是否正确，常常需要多次运用键盘运算。

(4) BASIC语言功能较强，不仅可适用于数值计算，而且还适合于数据处理，又能用于实时控制。BASIC的打印格式灵活多样，用来处理小型的事务管理任务是很方便的。BASIC还有绘图、音乐等功能，这是其它一些高级语言所不及的。因此，不少CAD（计算机辅助设计）、CAI（计算机辅助数学）和游戏程序是用BASIC语言编写的。

(5) 近年来，许多计算机所用的BASIC语言有很大的改进，已经出现了一些结构化的BASIC语言版本（如True BASIC，Quick BASIC，Turbo BASIC)，有的版本接近于结构化语言

(如 PC 机上用的 Microsoft BASIC，即 MS－BASIC)，便于进行结构化程序设计。

一些功能强的 BASIC 版本所提供的功能并不亚于 FORTRAN 语言，因此，BASIC 语言是有广泛群众基础又有实际使用价值的计算机语言。

由于 BASIC 采取解释执行方式，执行速度比较慢，这对完成较大型的复杂的任务是不利的，但对初学者所遇到的小型程序来说，解释执行利大于弊。现在已出现了一些采用编译方式的 BASIC（即编译 BASIC)，True BASIC 则解释方式与编译方式兼而有之，可以利用解释方式调试程序，编译方式运行程序，以达两全其美。

BASIC 提供的数据结构不如 PASCAL 和一些大型语言丰富，有些功能受到限制，因此，BASIC 一般用于相对小型的任务。

总之，在推广普及计算机应用中，BASIC 是一个有效的工具，尤其对非计算机专业的学生和科技人员、管理人员更为适宜。有了 BASIC 语言的基础，再学习其它计算机语言就比较容易了。不少同志在学习了 BASIC 后，消除了对计算机的神秘感，培养了对计算机的兴趣，掌握了程序设计的基本方法，在需要时再学习其它高级语言，能举一反三，不感到多大困难。

应当说明，尽管 BASIC 是一种国际上通用的计算机语言，但不同计算机系统上所使用的 BASIC 语言是有一些小的区别的。本书的叙述以在 IBM－PC，长城 0520，APPLE－Ⅱ（加 Z80 插件，并用 CP/M 操作系统)、王安 PC、ALTOS 等多种微机上都可运行的 Mucrosoft BASIC 为蓝本（简称 MS－BASIC)，也兼顾一般 BASIC 的特点，如果在读者所用的计算机上运行本书介绍的程序遇到困难，可查阅所用的计算机的 BASIC 手册，必要时对程序作少量修改即可。

## §2.2 BASIC 程序的构成和基本规则

先举例说明什么是 BASIC 程序和怎样运用 BASIC 语言算题。

求全班某课程的平均成绩：全班 32 人，A 等（5 分）13 人，B 等（4 分）12 人，C 等（3 分）5 人，D 等（2 分）2 人。

解此题可用 BASIC 语言编出下面的程序：

```
10  LET  A=13
20  LET  B=12
30  LET  C=5
40  LET  D=2
50  LET  TOTAL=A+B+C+D
60  LET  AVER=(5*A+4*B+3*C+2*D)/TOTAL
70  PRINT AVER
99  END
```

这种利用 BASIC 语言编制的算题程序，称 BASIC 语言程序，也称为 BASIC 语言源程序（Source program)。

上面程序中 A，B，C，D，TOTAL，AVER 是变量，A，B，C，D 分别代表 A 等、B 等、C 等、D 等的人数，TOTAL 代表“总人数”，AVER 代表“平均值”（取 Average 的前几个字母)。

BASIC 源程序有以下一些规定。

（一）一个 BASIC 程序（program）是由若干行（Line）组成的。一行内写一个或多个语句。如果在一行内写一个以上语句，则各语句间以冒号（：）分隔。例如，程序中的前四行也可以写成：

10 LET A=13：LET B=12：LET C=6：LET D=2

每一个语句分别让计算机执行某一个特定的功能。换言之，程序是由若干个语句构成的。例如，在上面的程序中，共包括 8 个语句。70 语句是让计算机打印出平均分数 AVER。

（二）每一行由（1）行号；（2）语句；（3）行结束符三部分组成。

每行前都冠以数字（在上例中是从 10 到 99），这个数字称为行号（Line number），如果一行只含一个语句则它就是语句标号，简称**标号**。行号必须是无符号整数，一般情况下，计算机会按行号大小顺序执行各行中的语句。行号的范围从 0 到 65529（不同的 BASIC 版本规定的范围不同），行号可不按大小顺序写，程序送入计算机后，BASIC 解释系统会把源程序中所有的语句行按行号大小顺序排列好，执行时依此顺序。行号不一定要连续（例如上例中两行间的行号间隔为 10），以便在修改程序时增补一些新的语句行。例如在行号为 10 的语句行和行号为 20 的语句行（为方便起见简称为 10 行和 20 行，如果该行只有一个语句，可简称为 10 语句和 20 语句）之间，根据需要最多可以插入 9 个语句行。

行结束符为“回车换行”，从键盘按“回车”（RETURN 键或 ENTER 键）表示本语句行到此结束。

（三）一个语句一般包括以下两个部分。

①语句定义符。它规定计算机执行某一特定的功能。例如上例中 10 语句 LET A=13，LET 就是一个语句定义符，意思是“赋值”，这个语句通知计算机把 13 这个值送到 A 中（或者说，把 13 这个值赋给变量 A）。除了赋值语句的语句定义符（LET）可以省写以外，一个语句如果没有语句定义符，就是错误的，计算机不能执行此语句，并打出错误信息。

②语句体，即跟在语句定义符后面的、需要执行的具体内容（例如 50 语句。需要进行的是将 A，B，C，D 的值相加，然后将它们的和送到变量 TOTAL 中去）。

在某些语句体中还包括行号，要进行语句处理（例如 GOTO 语句、GOSUB 语句等），我们以后就会看到的。

（四）每个程序一般以 END 结束。执行程序时，遇到 END 语句便停止执行。

（五）当一个程序通过输入设备（如终端显示器键盘）送入计算机后，就存在计算机内，计算机并不立即执行该程序。必须由使用者再单独发出“运行”的命令 RUN，计算机才开始执行该程序，一直执行到 END 语句，计算机就停止执行此程序。

例如上例中，在打入程序后，再打入 RUN 命令，则计算机开始执行程序，而按要求打印出 4.12499（即 AVER 的值）。

程序运行完后，并不从计算机中消失，如再一次打入 RUN 命令，计算机就使该程序再执行一遍（在上例中又打印出一个 4.12499 来），但是如果使用者又输入一个新的程序，而新程序的行号与原来程序的行号完全相同，则计算机就会清除原来的程序而以新的程序代替它，也就是说，如果以同一行号两次（或更多次）输入不同的语句，则计算机取最后一次输入的语句行。

（六）每一种计算机系统的 BASIC 分别规定了一个语句行最多包含多少个字符。例如 MS

BASIC 允许一个语句行可以容纳 255 个字符。当输入程序时字符超过显示器荧光屏上的一行（一般为 80 个字符）时，会自动转到荧光屏的下一行接着显示。但在计算机内仍把它们作为同一个语句行处理。有的 BASIC 则只允许在一行内写 80 个或更少的字符。

## §2.3 变　　量

程序中的变量是指在程序运行过程中其值可以变化的量。每一个变量在内存中占一定的存贮单元，在其中存放变量的值。可以形象地理解：每个变量为一个匣子，在匣子中放一个数据。例如程序中 10 语句“LET A=13”，意思是将 13 这个数据放到名字为 A 的变量中。见图 2.1。

一个变量在一个瞬时只能存放一个值。程序开始运行时 BASIC 自动使程序中每一个数值变量的初值为零。

| A | B | C | D |
|---|---|---|---|
| 13 | 12 | 5 | 2 |

图 2.1

每一个变量应有一个名字作标志。变量名的命名规定随不同的 BASIC 版本而异。MS BASIC 允许变量名长度不限，但只有前 40 个字符有效。变量名的第一个字符必须为字母，其后面的字符可以是字母、数字和小数点。如：A，B1，G1B2，SHANGHAI，E.JOHN 都是合法的变量名。而 LI-SUN，3AG4 为不合法。

有的 BASIC 版本对变量名限制比较严格，例如 Applesoft BASIC 只允许变量名最多由两个字符组成（第一个字符为字母），不允许出现小数点。

变量名应尽量按“见名知义”的原则命名，以便于理解其含义。如以 TOTAL 代表“总和”，以 AVERAGE 代表“平均值”，WAGES 代表“工资”，AMOUNT 代表“金额”、ALFA 代表 α，GAMA 代表 γ 等。

不应当用 BASIC 的保留字（包括所有的语句定义符、命令、函数名和运算符，见附录二）作变量名。如 SIN，LET，GOTO，END 都不能作变量名。

BASIC 变量分为数值变量和字符串变量两大类。前七章接触到的主要是数值变量，第八章介绍字符串变量。MS BASIC 的数值型变量还有整型、实型、双精度型之分，详见第十二章。

## §2.4 常　　量

BASIC 常量分为数值常量（即常数）和字符串常量（即字符串）。

数值常量（常数）有以下表示形式。

(1) 用日常的十进制数表示。例如：12，-34，376.5，-87.103 等。但应注意在一个数内不能用逗号分位，如 1000000 不能写成1,000,000。

(2) 用指数形式表示。在数学上常用指数来表示很大的数或很小的数，例如 12 亿可表示为 $1.2\times10^9$，蜗牛的速度 0.0000079 米/秒，可写成 $7.9\times10^{-6}$ 米/秒。由于键盘无法输入上角或下角，无法表示幂次，因此，改用字母 E 表示乘方的底数 10。如 $7.9\times10^{-6}$ 在 BASIC 中表示为：

7.9 E−06

数字部分　指数部分

下面是指数形式与十进制形式对应关系。

| **指数形式** | **代表的数** |
|---|---|
| 8.675E+03 | $8.675\times10^{3}$ |
| −7683.45E−12 | $-7683.45\times10^{-12}$ |
| 123.674E+14 | $123.674\times10^{14}$ |
| 6E7 | $6\times10^{7}$ |

一个数可以表示为多种不同的指数形式。例如：123.456 可以表示为 123.456E0，12.3456E1，1.23456E2，0.123456E3，1234.56E−1，12345.6E−2，123456E−3 等，都是合法的。BASIC 规定以数字部分有一位（且只有一位）非零整数的指数形式作为标准的指数形式（或称规范化的指数形式），例如上面各种指数形式中，只有 1.23456E2 为标准的指数形式。

对 BASIC 常数有两点要说明：

（1）数的范围是有限的，不同的 BASIC 版本规定的数的范围不同，如 MS BASIC 允许一个实数（以 X 表示）的范围为 $2.938736\times10^{-39}\leqslant|x|\leqslant1.701412\times10^{38}$。对绝对值小于下限 $2.938736\times10^{-39}$ 的数按 0 处理（称“下溢”），大于上限 $1.701412\times10^{38}$ 的都按上限值处理，并给出“溢出”信息（Overflow），称“上溢”。在一些 BASIC 中，在溢出的情况下程序仍可继续运行，但显然不能得到正确的结果。

（2）BASIC 所能接受和输出的有效数字位数是有限的，例如 MS BASIC 对实数（单精度数）可以接受和存贮 7 位数字，至多可以输出 7 位数字，但只有前 6 位数字是精确的（第 7 位有误差）。

在 BASIC 中数的范围和有效位数有限制，是由于在计算机内存中是以有限的内存单元来存贮一个数的。

需要说明：在计算机内部是以二进制数形式存贮和运算的，而在 BASIC 程序中是用十进制数形式表示的，用户不必自己将十进制数化成二进制形式，程序输入计算机后，由 BASIC 系统自动转换。

## §2.5　标 准 函 数

人们常常需要进行一些特定的运算，例如求 $\sin x$，$\cos x$，$\ln x$，$e^{x}$，$\sqrt{x}$ 等。为了方便使用者，BASIC 提供了一批标准函数，把常用到的一些函数运算编写成一个个子程序，组成函数库，用户在编程序时，不必自己查表或计算，只需写出一个函数名并给定自变量参数，BASIC 系统就能给出函数值。例如执行以下语句：

```
10    PRINT    SIN (3.14159/4)
```

会打印出 sin45°的值（在 BASIC 中三角函数中角度的单位一律为“弧度”而不是“度”。因此 π/4 弧度就是 45°。）

每种 BASIC 版本提供的标准函数有所不同，MS BASIC 的函数很丰富（见附录二），常用的函数及其含义如表2.1所示。

**标 准 函 数 表** **表 2. 1**

| 标准函数 | 功 能 | 备 注 |
|---|---|---|
| SIN (X) | sinx | 自变量单位以弧度表示 |
| COS (X) | cosx | 自变量单位以弧度表示 |
| TAN (X) | tanx | 自变量单位：弧度。注意：x 如接近 π/2（或 π/2 的奇数倍），则 tanx 可能会超过计算机允许的上限值而溢出 |
| ATN (X) | $\tan^{-1}x$ | 函数值的单位为弧度。−π/2<函数值<π/2 |
| LOG (X) | $\log_e x$（求以 e 为底数的对数值。即求自然对数 lnx） | x 值应为正 |
| EXP (X) | $e^x$ | e=2.718280。注意 x 的值不应使函数值超出计算机允许的范围 |
| SQR (X) | x 的平方根（正根） | x 应大于或等于 0 |
| ABS (X) | x 的绝对值 $\|x\|$ | |
| INT (X) | 求不大于 x 的最大整数 | 如：INT (−8.6)=−9， INT(8.6)=8 |
| FIX (X) | 将 x 截尾取整 | 如：FIX (−8.6)=−8， FIX(8.6)=8 |
| SGN (X) | 符号函数 | $SGN(x)=\begin{cases}1 & (当 x>0)\\0 & (当 x=0)\\-1 & (当 x<0)\end{cases}$ |
| RND [ (X)] | 产生（0，1）区间的一个均匀分布的随机数 | 一般 BASIC 规定 x 只有形式上的作用，可取任意值。MS BASIC 规定 RND（0）取（0，1）间的随机数，RND（1）得到上一次的随机数，RND（N）得到 1～N 间的随机整数 |

注意：自变量应该用括弧括起来，例如 SQR (4,5)，LOG (X) 是正确的，而 SQR4.5，LOGX 是不正确的函数表示，系统会把它们作为变量名处理。

# §2.6 表 达 式

## 2.6.1 算术运算符

所有 BASIC 都提供以下几种算术运算符：

＋（加），－（减），*（乘），/（除），∧（乘方）

如：$3\times5^2$ 表示为 3*5∧2。

MS BASIC 还提供：\（整除），MOD（求余）两种运算符。如：5\3=1（除数和被除数均为整数得到商也为整数），5 MOD 3 的值为 2。

## 2.6.2 算术表达式

在数学中，把用运算符号和括号将常数、变量、函数连接起来的有意义的式子称为代数表达式，如：Y－2＋SIN（X＋5）。同样，在 BASIC 中把符合 BASIC 规定的、用运算符和括号将常数、变量、函数连接起来的式子称为 BASIC 算术表达式，如：

5 * X ∧ 2 － 3 * X － 2 * SIN(A)/3

它是一个 BASIC 算术表达式，相当于通常的代数式：

$$5x^2 - 3x - \frac{2\sin A}{3}$$

应当特别注意的是：在 BASIC 的算术表达式中，每一个符号都占一格，所有符号都必须一个一个并排地写在同一横线上，不能在右上角或右下角写方次或下标（如 $2^2$，$X_2$ 等）。

下面列举一些代数式与 BASIC 算术表达式的对照：

表 2．2

| 项　目 | 代　数　式 | BASIC 算术表达式 |
|---|---|---|
| 圆周长 | $2\pi R$ | 2 * PI * R(其中 PI＝3.14159) |
| 欧姆定律 | $\frac{U}{R}$ | U/R |
| 等加速度求距离 | $V_{0t}+\frac{1}{2}at^2$ | V0 * T＋A * T ∧ 2/2 |
| 复利公式 | $P(1+r)^n$ | P *（1＋r）∧N |
| 三角形面积 | $\sqrt{s(s-a)(s-b)(s-c)}$ | SQR（S *（S－A）*（S－B）*（S－C) |

## 2.6.3 运算规则

先乘、除后加、减；乘方优先于乘除；函数优先于乘方；括号最优先。在 BASIC 中不分大、中、小括号，一律用（　），可以在（　）中套用（　），在套用括号多时应注意左右括号的配对关系。

用图 2.2 来表示优先次序：

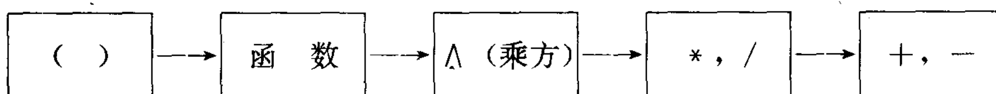
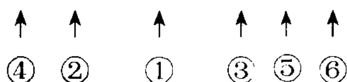



图 2.2

例如




小圆圈内的数字表示运算的顺序，也就是说先进行括弧内的运算，再进行函数 SIN 运算，求出 SIN（3 * X）后才进行求 5 次方的运算，乘和除是相同等级的，按先左后右运算，最后加 8。

如果有一数学式$\frac{a+b}{c+d}$，写成 BASIC 表达式的正确形式为（A+B）/（C+D），而不能写成 A+B/C+D。

## § 2.7 用传统流程图表示算法

在第一章中已说明：在编写程序之前，应当先确定数据结构和算法，对于一些简单的问题而言，数据结构是比较简单的，因此主要是设计算法，即具体描述解题的步骤。可以用自然语言来描述一个算法，但自然语言往往有"歧义性"，即同一说法可以有不同的理解，不严格。常用流程图来表示一个算法。如本章 § 2.2求平均成绩的例题，可以用图2.3流程图来表示解题步骤。

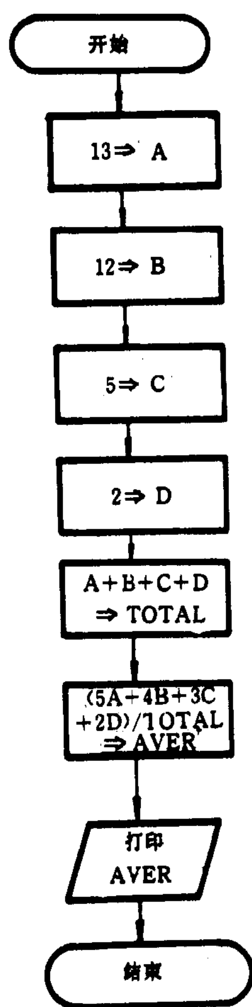



图 2.3

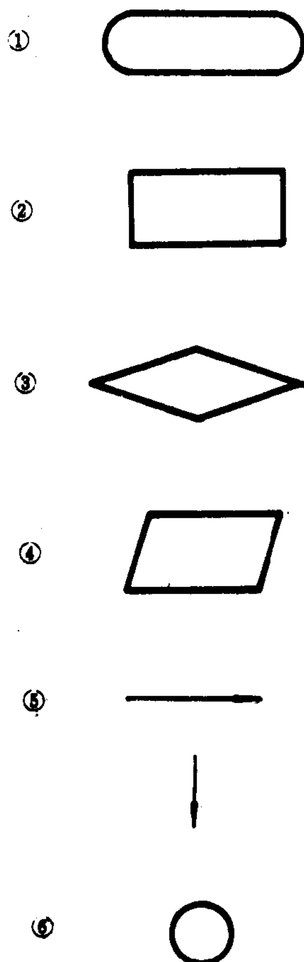



图 2.4

每一个框内写一个或几个操作，用箭头指出执行各框的顺序。可以看到，用流程图表示操作步骤形象易理解，用流程图可以帮助我们整理思路、表达算法、理解程序。对于简单的程序设计，可以不必先画流程图而直接写出程序（如上面例子），但对于比较复杂的题目，要一下子正确写出程序是不容易的，先要用流程图表示算法，由于流程图比较直观地表示出解题步骤，有错也容易发现，直到认为流程图完全正确才根据它来编程序。

常用的流程图符号如图2.4所示。

图2.4中的①是"起止框"，用来表示流程的开始或终止；②是"处理框"，用来表示一般的处理或运算，它有一个入口、一个出口，不能用作判断；③是"判断框"，有一个入口、二个出口，根据给定的条件是否满足决定选择其中一个出口路径；④是"输入输出框"，表示计算机输入或输出数据；⑤是"流程线"，用来表示流程的路径和

方向；⑥是"连接点"，表示将两个流程图中各自的某一点连接起来。当一个流程图在一页纸内画不下时可以用"连接点"表示从本页某一点（出口点）连接到下页某一点（入口点）。

**【例2.1】** 用流程图表示：求方程 $ax^2+bx+c=0$ 的根。

见图 2.5。为简单起见，如果 $b^2-4ac<0$，只打印出"复根"的信息，不具体计算复根。从图可以看到"判断框"的作用。在框内写一个条件，根据条件是否满足（以"是"、"yes"、"真"代表条件满足，以"否"、"NO"、"假"表示条件不满足）决定执行哪一种运算。

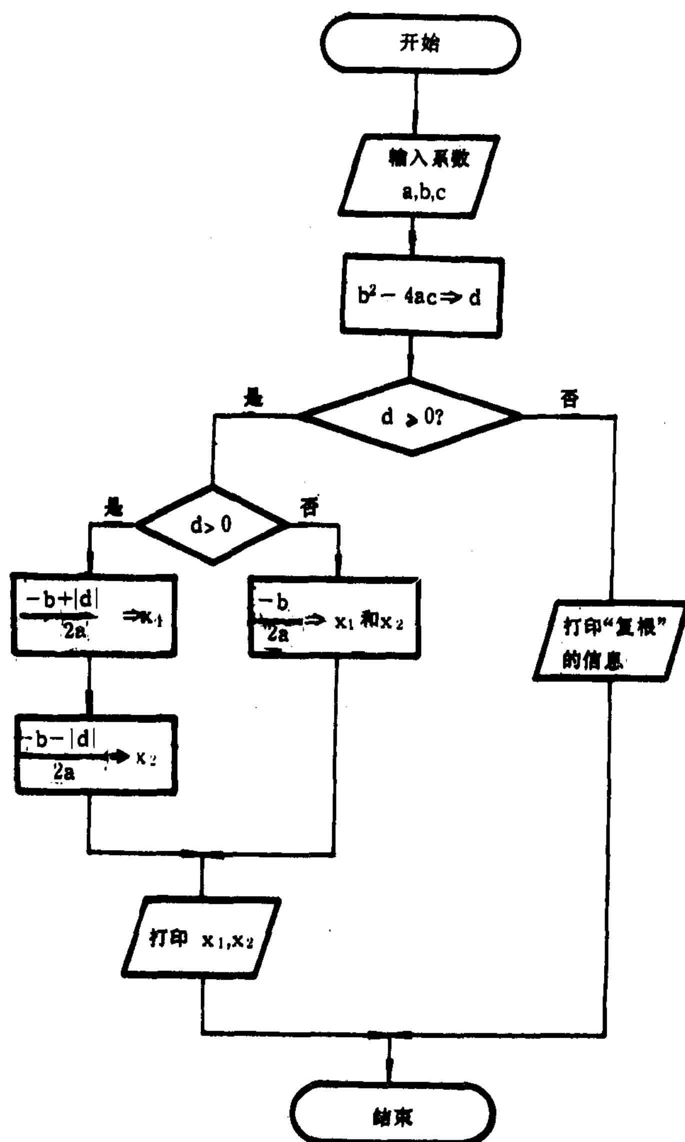



图2.5

**【例 2.2】** 从终端键盘不断输入数，把它们逐个累加起来（输入一个，累加一个，打印累加该数后的和），直到输入的数是负数为止。

设一变量 SUM 用来存放每次累加后的和（SUM 的初值自动为零），变量 $x$ 用来存放从键盘输入的值。每输入一个 $x$ 后，判断它是否大于或等于零，若是则将 $x$ 加到 SUM 中并打印此时 SUM 之值，然后再返回输入下一个 $x$ 值，重复以上操作，直到 $x<0$ 为止。见图2.6，可以看出，用流程图表示循环操作是很方便而又直观的。

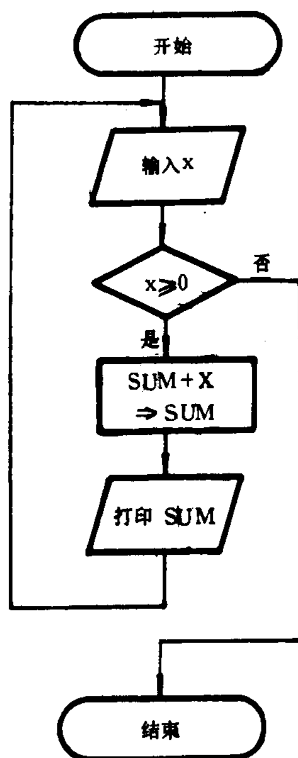



图 2.6



图 2.7

## §2.8 结构化程序设计要点和 N－S 结构化流程图

用上面介绍的流程图（称传统流程图）符号表示算法虽然直观，但占篇幅大，尤其是它允许用流程线任意转移去向，如果这种情况过多，就会使流程无规律地转来转去，如同一团乱麻一样，分不清其来龙去脉，见图 2. 7。这种程序称为 BS 型程序（BS 是 a Bowl of Spaghetti 的编写，意为"一碗面条"似的程序，搅成一团）。这种程序无任何规律可言，令人望而生畏，难以捉摸设计者的思路。因此，在初学程序设计时，应该养成一种良好的习惯。程序应当是深思熟虑、精心设计的结果，而不应当是随心所欲、东拼西凑的产物。无规律的转向太多就会造成程序设计和检查的困难，增加程序出错的可能，降低程序的质量。

为了提高程序的易读性（容易理解），保证程序质量，降低软件成本，荷兰学者 Dijkctra 等提出了"结构化程序设计方法"。它的要点主要是：

(1) 程序的质量标准是"清晰第一，效率第二"。而过去是把效率放在第一位，往往为了节约一点运行时间与内存空间，用了一些"小技巧"、"小聪明"，使人捉摸不透，程序涩晦难懂。现在把清晰放在第一位，因为程序是写给人家看的，在使用中经常要加以修改，如果要让人化大量时间才能看懂它，从整个社会所花的劳动量来看，是很大的浪费。

(2) 要求程序设计者按一定规范书写程序，而不能随心所欲地设计程序。过去的 BS 程序如同手工业方式，程序如同“艺术品”一样，风格因人而异。现在主张按照“工程化”生产方法来组织软件生产，每个人都必须按照同一规则，同一方法进行工作，使生产的软件有统一的标准、统一的风格，成为“标准产品”，便于推广、便于生产和维护。这就要找到一种标准化、规范化的产品式的程序设计方法。

(3) 结构化程序设计方法规定几种“基本结构”，用它们作为构成程序的基本单元，如同建筑房屋用的标准预制件一样。每一种基本结构完成一种类型的操作序列。由这些小单元顺序组成一个大结构，这种结构就避免了“任意转向”的缺点，是具有良好结构的程序。下面介绍的三种基本结构就属于这样的基本结构。

(4) 一个大的程序开发应当采取“自顶向下、逐步细化、和模块化”的方法。即将一个大任务先分成若干个子任务，每一个子任务就是一个模块，如果某一子任务还是太复杂，还可以再分解为若干个子任务，如此逐层分解。对每一个模块的设计也是采取这种“自顶向下、逐步细化”的方法，将它分解为上述的基本结构为止。这如同写文章一样，有人不订提纲，想到哪里写到哪里，而有人则先拟出总题目和中心内容，再确定分为几大部分，每一大部分又分为哪几节，每一节分为几段，每一段包括哪几个意思，有了这样通盘考虑的提纲后，文章的结构一般说是比较清楚的，考虑是周全的，不易发生遗漏。这就是自顶向下、逐步细化的方法。

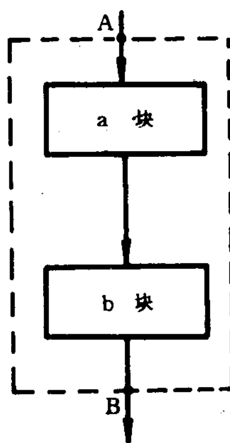



图 2.8

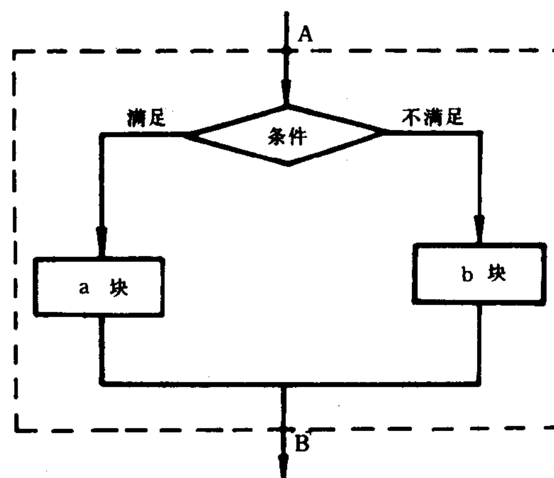



图 2.9

结构化程序的三种基本结构是：

(1) 顺序结构：见图2.8。这是最简单的一种结构。各块是按它们出现的先后顺序执行的。

(2) 选择结构，见图2.9。根据给定条件是否满足决定执行 a 块或 b 块。

(3) 循环结构，见图2.10和图2.11。

图2.10所示是“当型循环”，即“当给定条件满足时反复执行 a 块，直到条件不满足为止”。图2.11所示为“直到型循环”，即“反复执行 a 块，直到给定满足为止”。以上二者的区别是：“当型循环”是“先判断（条件）后执行（循环体 a 块)”，而“直到型循环”是“先执

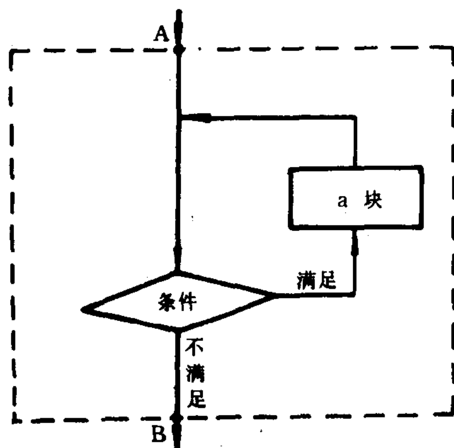



图 2.10

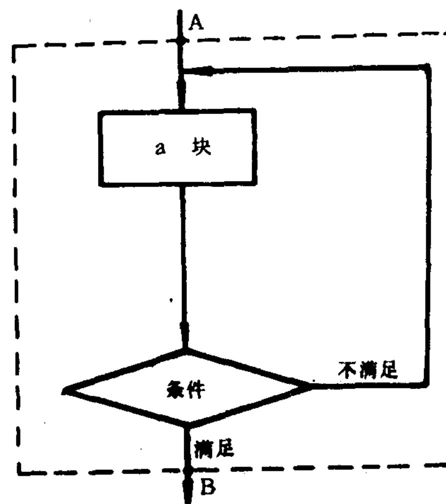



图 2.11

行（循环体）后判断（条件）”。可以认为，直到型循环是当型循环派生出来的一种循环形式，它们二者是可以互相转换的。　　图2.8～图2.11中的 a 块或 b 块，可以是一个简单的操作（一个语句），也可以又是一个基本结构，例如图 2.8 中的 a 块又可以是一个顺序结构。因此，图 2. 3 可以认为是一个大的顺序结构，它包含几个小的顺序结构。

以上三种基本结构具有以下共同特点。

(1) 只有一个入口、一个出口。注意是指上述各图中的虚线框（即整个结构）有一个入口一个出口，不要和虚线内的菱形框有两个出口相混淆。

(2) 通过该结构中的任一部分都应当存在着一条从入口到出口的路径，也就是说结构中每一部分都有机会被执行到，即没有“死语句”（永远执行不到的语句）。

(3) 没有“死循环”（即永远执行不完的循环）。

具有以上特点的结构称为“良结构”（或“合适结构”），结构化的程序是由一些良结构单元构成的。

既然结构化程序是由具有良结构特性的基本顺序组成，不允许无规律的转向，因此流程图中的流程线就可以省去了。根据美国学者 I. Nassi 和 B. Schneiderman 1973 年提出的方法的基础上形成了一种适于结构化程序设计的流程图，被称为“N－S 结构化流程图”或“N－S 流程图”，N、S 是他们二人姓名的首字母。它用以下三种基本成份表示三种基本结构。

(1) 顺序结构，见图2.12。

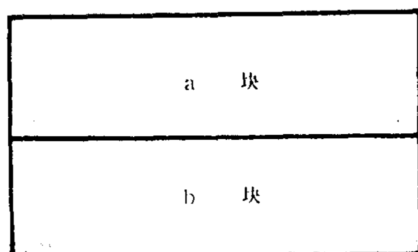



图 2.12

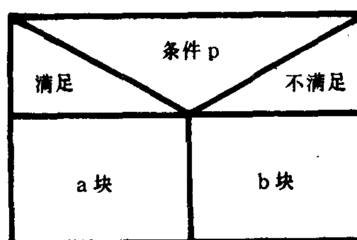



图 2.13

(2) 选择结构，见图2.13。

(3) 循环结构。图2.14为“当型循环”结构，图2.15为“直到型循环”结构。

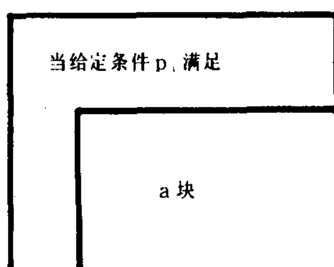



图 2.14

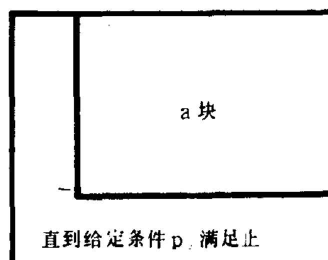



图 2.15

【例 2.3】用 N－S 流程图表示图 2.3 所示算法。

见图2.16。

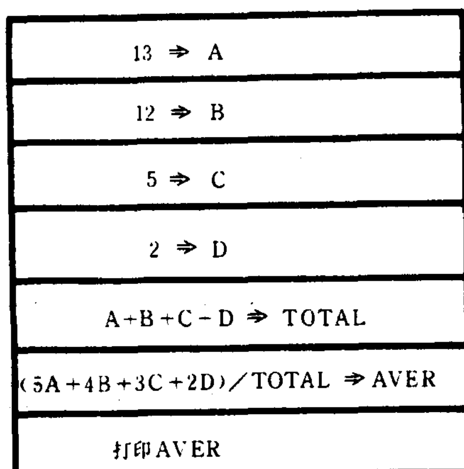



图 2.16

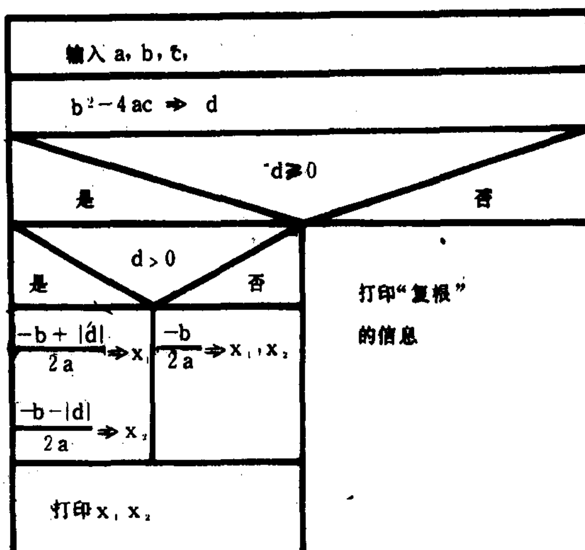



图 2.17

【例2.4】将例2.1的图2.5改用 N－S 图表示。见图2.17。

【例 2.5】将例 2.2 的图 2.6 改用 N－S 图表示。

图 2.6 不是结构化的，它不是由三种基本结构组成的，先对它作一些改换，使之成为由基本结构组成。图 2.18 与图 2.6 是等价的。图 2.18 中的虚线框是一个“当型循环”。图 2.18 是由三种基本结构组成的结构化的算法。可以用 N－S 图表示之，从图 2.19 可以看到，它是一个顺序结构，包括一个输入的操作和一个当型循环。

关于结构化程序设计方法在本章中只作初步介绍，在以后各章中逐步展开，读者将会看到如何设计一个结化的程序。对结构化程序设计方法欲进一步了解的可参阅本书参考资料 [2]。

最后要说明一点：流程图是画给人看的，而不是给计算机看的，因此图中的文字和符号只要符合人们的约定或习惯即可，人们据此写出的高级语言程序则是要输入给计算机处理的，语句必须严格符合语法规定。

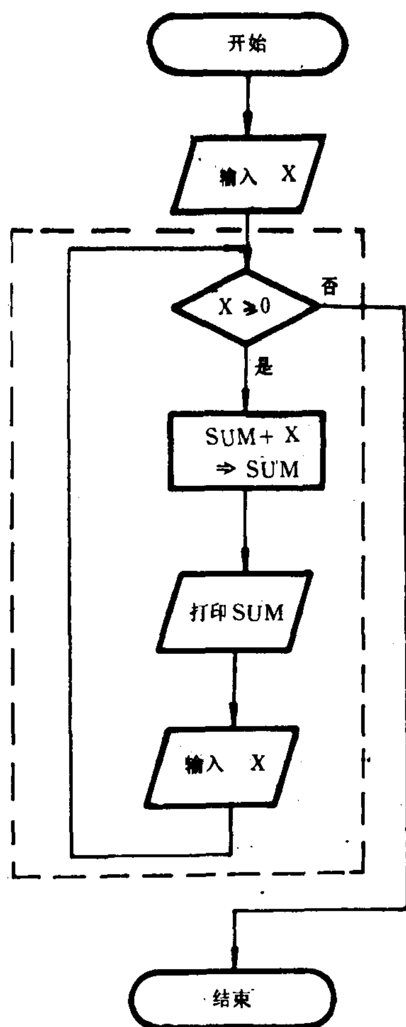



图 2.18

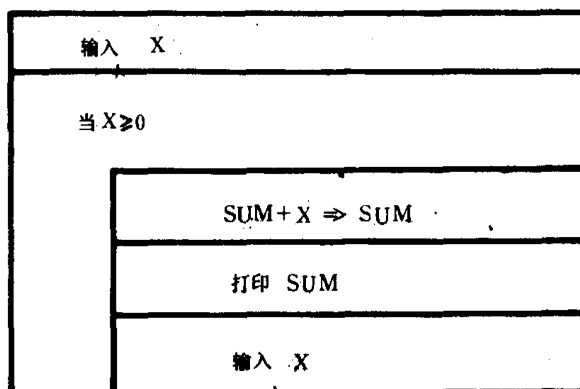



图 2. 19

## 习　　题

2.1　请分析 BASIC 语言的优缺点以及它的适用领域。了解和叙述你所用的 BASIC 语言版本的特点与功能。

2.2　BASIC 程序中的行号有什么作用？

2.3　BASIC 变量名的规则是什么？你用的 BASIC 版本对变量名有什么规定？请写出 5 个合法的变量名和 5 个不合法的变量名。

2.4　把下列各数由日常记数法转换成指数表示形式或由指数表示形式转换成日常记数法。

(1) 9423456　　(3) 0.9999999

(2) 7123456000000　　(4) 0.0000000004

(5) 8.67876E+8　　(7)3.1415928E+9

(6)0.123456E−6　　(8)2.567E−12

2.5　把下列代数式用 BASIC 表达式表示：

(1) $\frac{88 - 52 \times 63}{18 + 47 \div 3}$　(2) $\frac{(2\sin45°)^4}{e^2 \cdot \ln5}$　(3) $G \cdot \frac{m_1m_2}{r^2}$

分别用传统流程图和 N－S 图表示下列问题(2.6～2.10)求解的算法。

2.6　有两个变量 a,b,将它们的值互换。

2.7　送入计算机 10 个学生的成绩,要求将其中不及格的分数打印出来。

2.8　求 1+2+3+…+10,即 $\sum_{n=1}^{10} n$。

2.9　给出三角形三边 a,b,c,求三角形面积(要求先验算 a,b,c 是否符合构成三角形的条件)。

2.10　求方程 $ax^2 + bx + c = 0$ 的根(考虑 $d = b^2 - 4ac$ 等于零、小于零和大于零三种情况。如果 d<0,则分别打印出复根的实部和虚部)。

# 第三章　顺序结构程序设计

在本章中我们介绍几种简单的语句，并由它们组成顺序结构。这些语句包括赋值语句、输入输出语句、停语句，它们都是简单的操作语句，不具有控制流程的作用。

## §3.1　赋值语句(LET 语句)

赋值语句是将一个确定的值赋给一个变量。如：

```
10  LET  A=10
```

表示将 10 赋给变量 A，执行赋值语句后 A 的值为 10。

说明：

(1)赋值语句中的"="号是"赋值号"。赋值号的作用是将它右边的一个确定值赋给它左面的一个变量，赋值语句的一般形式为：

LET **变量=表达式**

例如：

```
10  LET  X=3*Y+1.5
```

执行过程是：①先计算表达式的值；②将该值赋给一变量。因此，赋值语句具有计算和赋值双重功能，先计算后赋值。BASIC 程序中的计算功能主要是由赋值语句完成的。如第二章§2.2 程序中求 TOTAL 和 AVER 的值都是由赋值语句完成的。

(2) 可以对一个变量多次赋值，每次赋值后新值将取代变量中原有的值，如：

```
10  LET  A=5

20  LET  A=10

30  LET  A=2*10
```

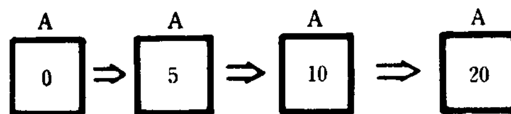



图 3.1

见图 3.1，在程序开始运行后，在执行 10 语句前 A 的值为 0，在执行 10 语句后，A 的值为 5，在执行 20 语句后 A 的值为 10，在执行 30 语句后，A 的值为 20。每一个瞬时一个变量只能有一个唯一的值。

(3) 不要把赋值号与数学中的等号混淆。等号表示其两侧的值相等，而赋值号的作用是将一个值赋给一个变量，下面两个语句的作用是不同的：

```
10    LET   A=B          (将 B 的值赋给 A)
10    LET   B=A          (将 A 的值赋给 B)
```

因此赋值号两侧的内容不能随意互换。如：

20 LET C=A+B

不能写成：

20 LET A+B=C

这从物理意义上也是很容易理解的：C的值不可能送到一个名为“A+B”的变量中去。赋值号左边只能是一个变量名，不能向一个表达式赋值。

在BASIC程序中常常用到像下面这种形式的赋值语句：

10 LET X=X+1

它表示将X的原值加1，把它们的和再送回变量X中。如果X原值为5，则X的新值为6。见图3. 2。

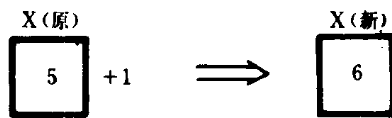



图 3.2

(4) 在大多数BASIC版本中，允许在赋值语句中省写语句定义符“LET”。下面两个语句是等价的。

10 LET A=B
10 A=B
}作用相同

但省写“LET”后，在思想上仍然要记住“这是赋值，不是相等”，千万不要误认为A等于B。

(5) 一个赋值语句只能给一个变量赋值，下面三个赋值语句的用法是错误的：

10 A=B=C=4 (原意想把4赋给A，B，C)

20 A，B，C=4 (想把4赋给A，B，C)

30 LET A=1，B=2，C=3 (想把1，2，3分别赋给A，B，C)

上面30语句应改为：

30 LET A=1：LET B=2：LET C=3

或 30 A=1：B=2：C=3

(6) 将一个变量A的值赋给另一变量B后，变量B取变量A之值，而变量A的值保持不变。不要误认为“A的值被取走了赋给B”。实际上是从内存中变量A的存贮单元中“读出”数值赋给B。可以从一个变量中多次“读出”数据进行运算或赋给其它变量。例如：

10 A=3：B=A：C=A：D=A

执行10行后，变量A、B、C、D的值均等于3。

## §3.2 输出语句(PRINT语句和LPRINT语句)

### 3.2.1 PRINT 语句的作用

程序运算得到的结果一般都要通过PRINT语句(或LPRINT语句)输出，使用户能够看到。几乎所有程序都包括PRINT语句。PRINT语句的作用是将变量或表达式的值输出到终端显示器的显示屏上。PRINT语句的一般格式如下：

PRINT **输出项表列**

(1) 输出数值表达式的值

PRINT 语句中的输出对项可以是数值常量、变量或表达式，例如：

```
10  PRINT  3, A, B, C+D
```

其实，常量、变量是表达式的一种形式。因此，可以这样说：PRINT 语句输出表达式的值。执行以下 10 语句的结果如下：

```
10  PRINT  (2+3)/2, 2*4,   5.5*4
RUN
 2.5                8                22
```

可以看到：PRINT 语句具有运算和输出双重功能，先进行运算，然后将值输出。如果在 PRINT 语句中的表达式中包括变量，则这些变量应当事先已被赋值，否则按零处理。如：

```
20  PRINT  X, X+Y, (C+D) /8
```

若事先未对以上各变量赋值，则运行结果为：

```
RUN
 0                 0                0
```

(2) 用 PRINT 语句除了可以输出数值外，还可以输出字符串。字符串是指用引号括起来的字符。在 PRINT 语句中如果包括字符串，则该字符串被原样输出。如：

```
10  PRINT  "THIS IS A BASIC PROGRAM."
20  END
RUN
THIS IS A BASIC PROGRAM.
```

可以看到字符串被原样输出，但应注意作为字符串起止符号的引号不输出。BASIC 中的引号是不分起始引号(“) 和终止引号 (”) 的，一律用双撇号 (″) 代替。(在本书中为印刷方便，仍用 “ 和 ” 符号。)

如果有以下语句：

```
10  PRINT "4+16"
```

运行时输出决不是 20，而是：

```
4+16
```

常用 PRINT 语句既输出数值表达式的值，又输出所需的字符串信息，如：

```
10  PRINT "4+16", 4+16
20  END
RUN
4+16=                  20
```

常用来对输出的数值作文字说明，如：

```
10  LET  X=3.14159*10∧2
20  PRINT  "X=", X
30  END
RUN
X=                     314.159
```

应当指出有些初学者容易混淆的问题：

(1) PRINT X，Y 这个语句的运行结果是打印出 X 和 Y 的值，而有些读者误以为会打印

出字母“X”和“Y”来。

（2）有的初学者写出如下形式的语句：

```
10  PRINT  X=3*5
```

原意是希望运行后能打印出 "X=15"。但这个语句的格式是错误的，PRINT 语句只能打印出表达式的值或字符串，而不能起赋值作用，因此，“=”号只能出现在字符串中，应改写为：

```
10  PRINT  "X=", 3*5
```

### 3.2.2 PRINT 语句的输出格式

程序计算的结果往往是一大批数据，为使输出结果清晰，人们往往对输出的格式有不同的要求，BASIC 的输出语句提供了多样的功能。

（1）按标准格式（标准位置）输出。

如果在 PRINT 语句中各输出项之间用逗号分隔，则输出时各值分别输出在各个标准位置上。BASIC 把每一行分为若干个区域，一般每个区为 14 列，一行包含 5 个区，各输出的值自左而右依次输出在各区内，从各区最左面开始输出。如：

```
10  PRINT  2, 4, 6, 8, 10
20  END
RUN
␣2␣            ␣4␣            ␣6␣            ␣8␣            ␣10␣
| 第1区        | 第2区        | 第3区        | 第4区        | 第5区        |
```

可以看到 2，4，6，8，10 分别出现在 5 个输出区内，在输出数值时，如果是正数则自动在数前留一空格，譬如第 1 区中，第一列为空格（以‘␣’表示空格），第 2 列是数值“2”，输出数值后，空一格，其它四个区情况相似。如果输出的是负数，则不留空格，在本区中第一列输出一个负号，如：

```
10  PRINT  -1, -2, -3
```

输出为：

```
|-2            |-4            |-6            |
|第1区         |第2区         |第3区         |
```

如果输出的是字符串，则从各区的第 1 列开始输出（没有符号位），如：

```
10  PRINT  "X", "Y", "Z"
```

输出为：

```
|X             |Y             |Z             |
|←第1区→       |←第2区→       |←第3区→       |
```

如果输出项超过 5 项，则在输出一行（5 个值后）自动换行到下一行继续按标准格式输出。

```
10  PRINT  1, 2, 3, 4, 5, -1, -2, -3, -4
```

输出为

```
␣1             ␣2             ␣3             ␣4             ␣5
-1             -2             -3             -4
```

（2）按紧凑格式输出。

如果在 PRINT 语句中各输出项间以分号分隔，则以紧凑格式输出（一个数据紧跟着前一个数据输出）。

```
10  PRINT  1; 2; 3; 4; 5; −1; −2; −3; −4; −5
20  END
RUN
␣1␣␣2␣␣3␣␣4␣␣5␣−1␣−2␣−3␣−4␣−5
```

请注意“1”和“2”之间有二个空格，其中一个是输出数值“1”后输出的空格，第二个空格是数“2”的符号位。在“−1”和“−2”间只有一个空格，它是输出数值−1 后输出的空格。

但是应当注意：字符串之间用分号分隔时，在输出时它们之间不留空格。

```
10  PRINT  "I"; "AM"; "A"; "PROGRAMMER"
20  END
RUN
IAMAPROGRAMMER
```

如果在 PRINT 语句中各输出项之间既有用逗号也有用分号分隔的，分别按标准格式和紧凑格式处理，凡遇逗号就跳到下一个区接着输出。

```
10  PRINT  1; 2; −3, 4; 5, 6, 7; −8; 9, 10
```

输出为：

| | | | | |
|---|---|---|---|---|
| ␣1␣␣2␣−3 | ␣4␣␣5 | ␣6 | ␣7␣−8␣␣9 | ␣10 |
| ←—第 1 区—→ | ←—第 2 区—→ | ←—第 3 区—→ | ←—第 4 区—→ | ←—第 5 区—→ |

（3）输出行的控制

①如果 PRINT 语句末尾无分号或逗号，则输出完本行后自动换行；

②如果 PRINT 语句末尾为分号，输出完本语句规定的各项后不换行，按紧凑格式在本行上接着输出下一个 PRINT 语句中第一个输出项（以后各项按规定接着输出）。

③如果 PRINT 语句以逗号结尾，输出完本语句规定的各项后不换行，在本行下一个输出区输出下一个 PRINT 语句的第一项（以后各项按规定输出）。

④PRINT 语句中无任何输出项，则输出一个空行，这称为“空 PRINT 语句”。当空 PRINT 语句前的一个 PRINT 语句最后带有一个逗号或分号时，此空 PRINT 语句只是起到抵消前面那个逗号或分号的作用。例如：

```
10  PRINT   1, 2, 3;
20  PRINT   4; 5,
30  PRINT   6,
40  PRINT
50  PRINT   7, 8,
60  PRINT
70  PRINT   9
80  END
RUN
```

```
␣1              ␣2              ␣3␣␣␣4␣␣␣5      ␣6
␣7              ␣8
␣9
```

请注意 4 紧连在 3 之后，6 与 5 同一行（6 在下一个输出区），如果无 40 语句，则 7 应与 6 在同一行上，由于有了 40 语句使之换行，所以 7 输出在下一行上，它的作用是抵消了 30 语句末尾的分号。60 语句作用相似。

应当说明，不同的 BASIC 版本对输出格式有不同的规定，例如有的规定一行只有四个标准输出区，有的（如 Applesoft BASIC）规定在输出数值时既不留符号位（对正数）又不留尾随空格。如：

```
10  PRINT  1; 2; -3; -4
```

输出为：

```
12-3-4
```

显然这种格式是不好的，数据间无清晰的分界线。

除了以上介绍的以外，BASIC 还提供其它输出格式（如 TAB 函数、SPC 函数、PRINT USING 语句等），详见第六章和第十章。

归纳起来，PRINT 语句的一般格式为：

$$\text{PRINT}\quad [\text{表达式表列}]\left[\left\{\begin{matrix},\\;\end{matrix}\right\}\right]$$

其中方括弧表示其内是可选项（该项可有可无），花括弧表示从其中几项中任取一项，$\left[\left\{\begin{matrix},\\;\end{matrix}\right\}\right]$的意思是：有一个“,”或有一个“;”，或二项都没有。

### 3.2.3 实数的输出形式

一个实数在内存中是以二进制的指数形式存储的，在输出时，实数有时以小数形式输出，有时以指数形式输出。例如：

| 实数值 | 输出形式 |
|---|---|
| 123.456 | 123.456 |
| -0.025 | -0.025 |
| 0.00001 | 0.00001 |
| 0.00000123456 | 1.23456E-06 |
| -12345689 | -1.234569E+06 |

它的原则是：当以 7 位或少于 7 位数字能精确地表示出该数时，就以小数形式输出（如上表中第 1～3 行），否则就以指数形式输出（上表第 4、5 行）。0.00000123456这个数如果用小数形式输出只能得到 7 位数字，即0.0000012，把有用数字“23456”丢失了。第 5 行“-12345689”有 8 位数字，超过了系统允许的有效位数，无法表示，改用指数形式才能较准确地表达该数，对第 8 位按四舍五入处理，得-1.234569。以上判断处理是由 BASIC 系统自动

进行的。在按指数形式输出时，一律采用标准化的指数形式（如 1.23456E—06，—1.234569E+06，小数点前有且只有一位非零的数字）。

用指数形式输出是为了保证能输出七位有用数字。应当说明：不同的 BASIC 版本在实现时有不同的规定。

### 3.2.4 LPRINT 语句

用 PRINT 语句可以将输出信息输出到显示器荧光屏上，MS BASIC 还提供一种在打印机输出的语句——LPRINT 语句。它的功能和 PRINT 完全相同，只是不在显示器上输出而在打印机上输出。例如：

```
10  LPRINT    1; 2, 3
```

在打印机上输出

```
1␣␣2            3
```

在程序中同时使用 PRINT 语句和 LPRINT 语句，可以根据需要将一部分信息输出到显示屏上面将另一部分信息输出到打印机。但是不是所有的 BASIC 版本都提供 LPRINT 语句的。

## §3.3 键盘输入语句（INPUT 语句）

INPUT 语句用于从键盘输入值给变量，也就是说，在程序清单中是看不到变量的值的，变量的值是在程序运行过程中临时由用户从键盘输入的。例如：

```
10  INPUT  X
20  PRINT  X*X
30  END
RUN
? 3  ↙
 9
```

计算机执行到 INPUT 语句时，就停下来显示出一个“?”，意思是向使用者询问 INPUT 语句中的变量的值。使用者应该从键盘上输入所需的值，然后按回车键，于是这个数值就赋给相应的变量，接着计算机再继续执行后面的语句。在上面程序运行记录中，3 和回车↙是使用者从键盘打入的（↙这个符号并不在终端显示器上显示出来），9 是程序执行到 20 语句时显示出来的值。

用一个 INPUT 语句，可以输入多个变量的值。

用 INPUT 语句的好处在于可以根据需要在多次运行程序时给一个变量赋予不同的值，以提高程序的通用性。

【例 3.1】有一电路（见图 3. 3)，求并联电阻 R 和电流 I。

求并联电阻的公式为：

$$\frac{1}{R}=\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}+\frac{1}{R_4}$$

电流 I 为：

$$I=U/R=U\left(\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}+\frac{1}{R_4}\right)$$

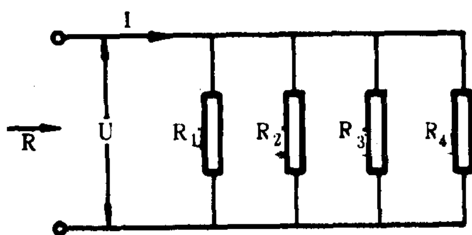



图 3. 3

如果用赋值语句给 $R_1$，$R_2$，$R_3$，$R_4$ 和 U 赋值，程序如下：

```
10  LET  R1=100
20  LET  R2=150
30  LET  R3=240
40  LET  R4=360
50  LET  U=24
60  LET  R=1/ (1/R1+1/R2+1/R3+1/R4)
70  LET  I=U/R
80  PRINT  R, I
99  END
```

如果电路中的参数改变了，例如：$R_1=50$，$R_2=100$，$R_3=500$，$R_4=1000$，U=220，那么需要重写 10～50 语句。若要计算多个具有不同阻值的电路，则需多次修改程序，显然是十分不方便的。可以用 INPUT 语句方便地解决这个问题。

```
10  INPUT  R1, R2, R3, R4, U
20  R=1/ (1/R1+1/R2+1/R3+1/R4)
30  I=U/R
40  PRINT  "R="; R, "I="; I
50  END
```

在显示屏上看到的运行情况如下：

```
RUN
? 100, 150, 240, 360, 24↙
R=42.35565        I=.5666667
```

如果阻值和电压改变了，需要重新计算，不必修改程序，只需再运行一次，输入不同的值即可。如：

```
RUN
? 50, 100, 500, 100, 220↙
R=23.80953        I=9.239999
```

显然这是很方便的。为了使用户知道："现在应该输入哪几个变量的值"，可以在执行 INPUT 语句时让系统显示用户所指定的"提示信息"。可以在 INPUT 语句中加入"提示信息"部分。如：

```
10  INPUT  "INPUT  R1, R2, R3, R4, U="; R1, R2, R3, R4, U
```

引号中就是用户指定的"提示信息"。

程序运行记录如下：

R1，R2，R3，R4，U=？ 100，150，240，360，24↙

R=42.35295　　　　I=.5666667

其中带下划线的部分是用户从键盘输入的。这个10语句相当于下面两个语句：

5 PRINT "INPUT R1，R2，R3，R4，U=";

10 INPUT R1，R2，R3，R4，U

INPUT 语句的一般格式为：

INPUT [ "提示信息";] 变量 [，变量] …

其中省略号"…"表示它前面的项(即[ ]内的内容)可以出现多次(如 INPUT A,B,C)。

如果想不输出提示信息后面的"?",可以将 INPUT 语句中"提示信息"后面的分号改为逗号。如：

10 INPUT "INPUT R1,R2,R3,R4,U=",R1,R2,R3,R4,V

执行此语句的记录如下：

INPUT R1,R2,R3,R4,U=100,150,240,360,24↙

使用 INPUT 语句时有几点应注意：

(1)输入数值的个数应与 INPUT 语句中变量的个数一致,数据输入后按次序输给相应的变量。如果输入数据的个数多于或少于变量的个数,BASIC 系统会显示出一个错误信息:"?REDO FROM START。"要求用户重新输入所需数据。(不同的 BASIC 系统对此处理方法不同,有的 BASIC 对输入的多余的数据认为无效,如果数据个数少于变量个数,则在下一行又显示出一个"?"或"??",要求用户继续输入其它的数据,直到满足要求为止)。

(2)用 INPUT 语句输入的数据只能是常数,不允许是表达式或变量、函数。如下面的输入是错误的。

INPUT R1,R2,R3,R4,U=? 2*50,600/3,20∧2,B,SQRT(460,5)↙

INPUT 语句常用于参数变化、要进行不同参数的运算结果比较的情况。在设计中,常常需要使一个或几个参数变化,并观察其计算结果,这时可利用 INPUT 语句,边计算边修改参数,直到得到满意的结果为止。

(3)INPUT 语句的输出和输入是可以在显示屏上看得见的,如果使用长城(或 IBM PC)的 MS BASIC(即 BASICA),在打印机上只能得到 LPRINT 语句所输出的结果而不能得到 INPUT 语句所处理的输出和输入情况,此时可以在程序中另加一个 LPRINT 语句将所需数据打印出来,例如例 3.1 程序可以改为：

```
10 INPUT "INPUT R1,R2,R3,R4, U=";R1,R2,R3,R4,U
15 LPRINT "INPUT R1,R2,R3,R4,U=?";R1;",";R2;",";R3;",";R4;",";U
20 R=1/(1/R1+1/R2+1/R3+1/R4)
30 I=U/R
40 LPRINT "R=";R,"I=";I
50 END
```

在运行时在打印机输出以下信息：

INPUT R1,R2,R3,,R4,U=? 100,150,240,360,24(由15语句输出)

R= 42.35332　　　I= .5666615　　　(由40语句输出)

此外，在显示屏上显示：

```
INPUT  R1, R2, R3, R4, U=?   100, 150, 240, 360, 24
              ↑                          ↑
      10 语句输出的字符串            从显示器键盘输入
```

注意，用 LPRINT 语句只能在打印机上输出，而不能在显示屏输出。

## §3.4 读数语句(READ 语句)和置数语句(DATA 语句)

除了可以用赋值语句和 INPUT 语句给程序中的变量赋值之外，还可以用 READ 语句和 DATA 语句使变量得到值。例如对例 3. 1 可以用 READ/DATA 语句编写程序如下：

**【例 3.2】** 求并联电阻 R 和电流 I

```
10  READ R1, R2, R3, R4, U
20  DATA 100, 150, 240, 360, 24
30  R=1/ (1/R1+1/R2+1/R3+1/R4)
40  I=U/R
50  PRINT "R=", R, "I=", I
60  END
```

运行结果与以前相同，说明如下：

(1) READ 是读数语句，它是执行语句，它的作用是从程序中的 DATA 语句中取数据依次送给 READ 语句中的变量。上例中，将 100 送给 R1，150⇒R2，240⇒R3，360⇒R4，24⇒U。因此，程序中如果有 READ 语句则必须有 DATA 语句。

(2) DATA 语句是非执行语句，它本身不产生任何操作，它的作用只是放置数据以供 READ 语句使用。因此 DATA 语句中的数据个数应不少于 READ 语句所需的变量个数(DATA 语句中数据个数如多于 READ 语句所需个数，多余的数据不起作用)。DATA 语句有如放货物的仓库，READ 语句相当“取货员”，根据领货单从仓库中取出货物。DATA 语句中各个数据间以逗号分隔。

(3) DATA 语句可以出现在程序中任何位置，可以在 READ 语句之前，也可以在其后，可以与 READ 语句相邻也可以不相邻。习惯上把 DATA 语句集中放在程序最前头或程序最后(END 语句之前)。效果完全相同。正如有一书架放书，只要书架中的书次序不变，不论书架搬到室内任何角落，找到书架后，拿到的第一本书是不会变的。

(4) 一个程序中可以有多个 DATA 语句。在程序输入计算机后，BASIC 在内存建立一个数据区，将 DATA 语句中的数据按出现的顺序放到数据区内。如：

```
10   READ  A, B, C, D, E, F, G
 ⋮
120  DATA  1, 2
130  DATA  3, 4, 5, 6
140  DATA  7, 8, 9, 10
```

数据区见图 3. 4 示意。如果将 120 行的行号改成为 5，使 DATA 语句分隔开，在内存中数据状况不变，仍如图 3. 4 示。

可以认为数据区有一“数据指针”，开始时指向第一个数据（图中①位置），当读完第一

个数据(赋给变量 A)后,指针自动移动指向下一个数据(见②位置)。以后每读一个数据指针下移一个位置,当执行完 READ 语句后,指针位置如③。

如果程序中有多个 READ 语句,则依 READ 语句中变量出现的顺序依次从 DATA 语句"取数"。第一个 READ 语句中的变量从程序中行号最小的 DATA 语句中的第一个数据开始读数。当第一个 DATA 语句中的数已被读完,指针就移到第二个 DATA 语句的第一个数据前,并接着往下读,如果第一个 DATA 语句中的数据多于第一个 READ 语句中变量的个数,则第二个 READ 语句中的变量接着读第一个 DATA 语句中未被读过的数据,依此类推。(从图 3. 4 看这个过程是很清楚的,因为变量从数据区中顺序读数)

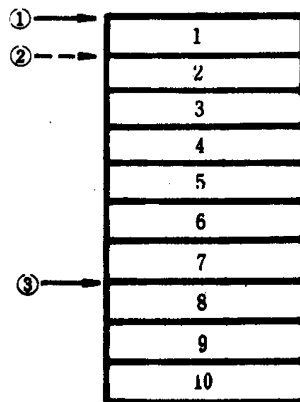



图 3. 4

(5)DATA 语句中只能出现常量,不能出现表达式或变量、函数。下面写法是不合法的:

```
100  DATA  5/2, 2*3, SIN(6.5)
```

(6) 如果所有的 DATA 语句中数据的个数比 READ 语句所需的数量少,就会出现"数据区无数据可读"的错误,系统显示出"Out of data"的错误。

(7) READ 语句和 DATA 语句的一般格式为:

READ **变量** 〔, **变量**〕…

DATA **常量** 〔, **常量**〕…

## §3.5 恢复数据区语句(RESTORE 语句)

如果 DATA 语句中的数据是重复的或一部分是重复的,可以用 RESTORE 语句重复使用 DATA 语句中的全部或部分数据。例如:

```
10  READ A, B, C, D, E
20  READ M, N, O, P, Q
30  READ V, W, X, Y, Z
 ⋮
80   DATA 1, 2, 3, 4, 5
90   DATA 1, 2, 3, 4, 5
100  DATA 1, 2, 3, 4, 5
```

80 语句到 100 语句中的数据是完全重复的,可以设想,在读完第一组数据 (1, 2, 3, 4, 5) 后如果能使数据区指针恢复到起始位置,那么这组数据又可以再使用一遍。RESTORE 语句的作用就是使数据指针恢复到起始位置(或指定的其它位置)。程序可改为:

```
10  READ A, B, C, D, E
15  RESTORE
20  READ M, N, O, P, Q
25  RESTORE
```

```
30  READ V, W, X, Y, Z
  ⋮
100  DATA 1, 2, 3, 4, 5
```

在执行 10 语句前，数据指针为图 3. 5 中①位置，如果无 15 语句，则执行 10 语句后，DATA语句中数已读完，指针位置如图中②位置，本来已无数可读。有了 15 语句，使指针重新回到 DATA 语句中的第一个数据（即数值 1）之前（即①位置），执行 20 语句时，又从 1 起读数。25 语句的作用与 15 语句相同。

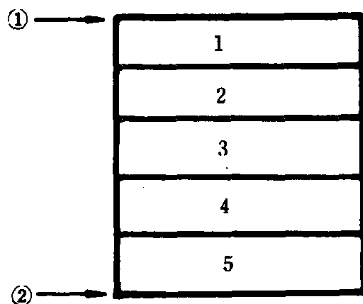



图 3. 5

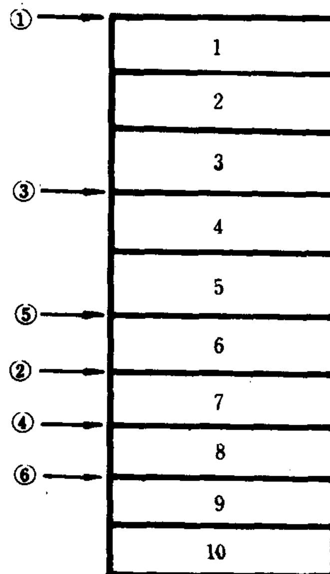



图 3. 6

当重复利用的数据愈多，RESTORE 语句的优越性就愈明显。

如果只需要重复使用一部分数据，可以在 RESTORE 语句中指出行号，如：

```
10  READ  A, B, C, D, E, F
20  RESTORE  110
30  READ  G, H, I, J, K
40  RESTORE  120
50  READ  L, M, N
  ⋮
100  DATA  1, 2, 3
110  DATA  4, 5
120  DATA  6, 7, 8, 9, 10
```

开始时指针位置为图 3. 6 中①所示，执行完 10 语句后指针位置如②所示。执行 20 语句后指针位置如③所示（指针恢复到 110 语句中第一个数据处），执行完 30 语句后指针位置为④示，执行 40 语句后，指针位置如⑤示，执行完 50 语句后指针位置如⑥示。此时各变量的值如表 3. 1。

表 3.1

| A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 | 6 | 7 | 8 |

RESTORE 语句的一般格式为

RESTORE 〔行号〕

要注意的是：RESTORE 的作用是使 DATA 语句中的数据从头或指定的位置开始，而不是使程序回到第一个 READ 语句，然后又从 READ 语句中的第一个变量重新读起。这一点初学者常常容易弄错。

## §3.6 三种提供数据的语句比较

在 BASIC 中可以用 LET 语句、INPUT 语句和 READ/DATA 语句给变量提供所需的数据。这三种语句各有特点。LET 语句除了用来给变量赋值之外，它的主要优点是能够进行运算。如果要赋值的变量较多，则用 READ/DATA 语句比较方便。INPUT 语句适合于变量的值事先未确定或参数值有变化需要进行试验比较的情况。具体比较见表 3.2。

表 3.2

| LET | READ/DATA | INPUT |
|---|---|---|
| 1. 三种语句都能给程序中的变量赋值 | | |
| 2. 每一个 LET 语句只能给一个变量赋值 | 一对 READ/DATA 语句可以给多个变量赋值 | 一个 INPUT 语句可以给多个变量赋值 |
| 3. 如要赋值的变量多，则程序较长 | 程序较短 | 程序短 |
| 4. 在一个程序中用 LET 语句只能给一组变量赋值(不能用 GOTO 语句或循环语句使一组变量获不同的值) | 在一个程序中可以给一组变量多次赋值(用 GOTO 语句或循环语句使变量多次读数) | 在一个程序中可以多次从键盘输入不同的数值给同一变量（用 GOTO 语句或循环语句使程序多次读数） |
| 5. 运行后立即出结果占机器时间少 | 运行后立即出结果，占机器时间少 | 运行后要等待键盘输入，占机器时间多 |
| 6. 可以进行运算(即可用表达式给变量赋值) | 不可以进行运算。DATA 语句中只能写常数，不能写表达式 | 不可以进行运算。只能从键盘输入常数，不能输入表达式 |
| 7. 适合于①需要赋值的变量少、不需反复计算和比较简单的程序。②进行运算必须用 LET 语句 | 适合于输入数据较多(而且都有确定的数值）的程序 | 适合于参数变化、需要进行多组数据运算和比较的程序 |

# § 3.7 END 语句和 STOP 语句

## 3.7.1 END 语句

一个程序应当有一个 END 语句，以终止程序的执行，一般它出现在程序最后一行。MS BASIC 也允许它出现在程序中任何位置，而有的 BASIC 要求程序最后一行必须是 END 语句。我们建议应养成程序最后写 END 语句的习惯，这样符合顺序阅读程序的习惯，最后一行完了程序就结束。

## 3.7.2 STOP 语句和 CONT 命令

STOP 也用来终止程序的执行并返回 BASIC 状态，在执行 STOP 语句时，会显示出以下信息：

Break in nnnnn

其中 nnnnn 代表行号的数字，它是 STOP 语句的行号。在程序遇到 STOP 语句中断运行后，可以用 CONT 命令使之从中断处恢复运行，也可以用键盘命令打印出当时（暂停后）各有关变量的值。

**【例 3.3】** 输入三角形三边 A，B，C，求三角形面积。

```
10  INPUT  A, B, C
20  PRINT  A+B-C, B+C-A, C+A-B
30  STOP
40  S= (A+B+C) /2
50  AREA=SQRT (S* (S-A) * (S-B) * (S-C))
60  PRINT  AREA
99  END
```

在输入 A，B，C 后，要检查一下它们是否能构成三角形（三角形二边之和应大于第三边），20 语句显示出三个值，执行 30 语句使程序中断，显示："Break in 30"，用户检查这三个值是否大于零，如都大于零，表示数据合适，可以接着计算三角形面积，此时可从键盘打入以下命令：

CONT↙

使程序从中断处恢复运行（CONT 是 Continue 的缩写）。如果经检查发现有一个以上的值不大于零，则不应计算下去，而应当重新运行，输入另一组 A，B，C 的值，也可以用 GOTO 10 使之回到 10 语句重新执行。

在遇 STOP 语句程序中断后，各变量的值和 DATA 语句中的数据指针都保留暂停前的情况，例如变量 A，B，C 的值仍保持不变，可以在中断后用 PRINT 语句显示出程序中各变量当时之值。也可以用键盘命令直接改变变量的值，例如：

```
LET  A=8
LET  B=6
LET  C=4
```

然后用"GOTO 20"使从 20 语句处继续执行。

常用 STOP 语句和 CONT 命令来进行程序的调试，比较方便。但如果在中断期间对程序作了修改，就不能再用 CONT 命令。

STOP 语句和 END 语句不同，STOP 语句不关闭文件，而 END 语句则关闭程序中所有文件（有关文件的概念见第十一章）。

## §3.8 程 序 举 例

到目前为止，我们已能够用所介绍的语句进行简单的程序设计了。

**【例3.4】** 人造卫星围绕地球表面做近似圆周运动，卫星轨道距地面 $h$ 米高时的圆周运动速度 $V_c$（米/秒）为：

$$V_c=\frac{7900\sqrt{R}}{\sqrt{R+h}}$$

$R$ 是地球半径≈$6.37154\times10^6$ 米。在某一特定高度上，卫星脱离轨道的速度为：

$$V_e=V_c\sqrt{2}$$

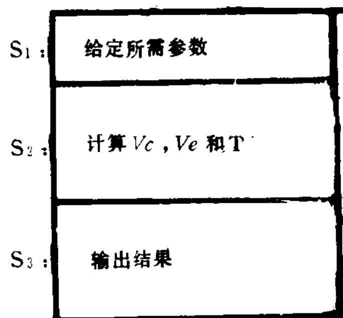



图 3. 7

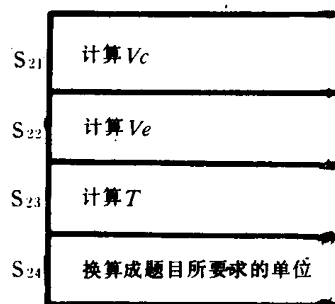



图 3. 8

今要求给定一个 h 后，计算卫星的圆周速度和脱离轨道的速度（用千米/小时单位）以及环绕地球一周所需要的时间 T（用分表示）。

对这样比较简单的问题可以很快地画出顶层的 N－S 流程图。（图 3. 7）。$S_1$ 和 $S_3$ 两步是比较简单的，无需细化。对 $S_2$ 可以进一步细化如图3.8。至此，可以根据流程图写出程序。

```
10  R=6.37154E6   （地球半径）
20  INPUT "ENTER HEIGHT:", H
30  VC=7900*SQR (R)/SQR(H+R)
40  VE=VC*SQR (2)
50  C=2*3.14159*(H+R)     （飞行一周的距离）
60  T=C/VC             （飞行一周的时间，单位：秒）
70  D=3600/1000          （化成千米/小时单位应乘的系数）
80  VC=VC*D: VE=VE*D    （化成千米/小时单位）
90  T/60          （化成分）
```

```
100  PRINT "VC="; VC; "KM/HOUR"
110  PRINT "VE="; VE; "KM/HOUR"
120  PRINT "TIME="; T; "MINUTES"
999  END
RUN
ENTER  HEIGHT: 20000↙
VC= 28395.47 KM/HOUR
VE= 40157.26 KM/HOUR
TIME= 84.85691 MINUTES
```

【例3.5】 分期付款的计算。如果你从银行借了一笔款 $D$，已知你准备每月偿还的款额 $P$ 和月利率 $R$ 求需要多少月（$M$）才能还清。

计算还完此款所需多少月的公式为：

$$M = \frac{\log P - \log(P - D \times R)}{\log(1 + R)}$$

可直接写出程序：

```
10  INPUT "LOAN  LIMITS:"; D
20  INPUT "INSTALLMENT:"; P
30  INPUT "INTEREST:"; R
40  K=P/(P-D*R)
50  A=LOG(K)/LOG (10): B=LOG (1+R) /LOG (10)
60  M=A/B
70  PRINT "LOAN LIMITS:", "", D
80  PRINT "INSTALLMENT:", "", P
90  PRINT "NUMBER OF INTEREST:", "", R
100  PRINT "NUMBER OF INSTALLMENT:", M
999  END
```

BASIC 中函数 LOG (X) 代表 $\ln x$（以 $e$ 为底），而不代表 $\log_{10} x$。因此求 $\log_{10} x$ 应写成表达式 LOG (X) /LOG (10)，50 行第一个语句求的 $A$ 是 $\log_{10}$K，即 $\log_{10}(\frac{P}{P-D*R})$，也就是 $\log_{10} P - \log_{10}(P - D \times R)$。

运行记录为：

```
RUN
LOAN LIMITS: 2000↙        (借款额)
INSTALLMENT: 100↙         (每月归还数)
INTEREST: .012            (月利率)
LOAN LIMITS:                          2000
INSTALLMENT:                          100
INTEREST:                             .012
NUMBER OF INSTALLMENT:                23.00668   (还清期限)
```

请读者对照程序和运行结果，分析 70～90 语句中的第二对引号（"　",）的作用。

如果想对得到的结果进行四舍五入处理（如23.00668月应输出 23 个月），有什么办法吗？

下面介绍一种方法：对一个数 $X$ 四舍五入的方法是：Y=INT（$X$+0.5），如果：

$X$=24.49，则 $X$+0.5=24.99，INT（$X$+0.5）=24。

$X$=24.51 则 $X$+0.5=25.01，INT（$X$+0.5）=25。

四舍五入的实现方法请读者仔细理解清楚。

若希望输出月数为整数（四舍五入）可以将 60 语句改为：

60 M=INT (A/B+0.5)

**【例3.6】** 鸡兔同笼，已知鸡兔总头数为 $H$ (Heads)，总脚数为 $F$ (Feet)，求鸡兔各有多少只？解此题的步骤是：

(1) 先列出解此问题的方程式。设鸡为 $X$ 只，兔 $Y$ 只。

$$\begin{cases} X + Y = H & (1) \\ 2X + 4Y = F & (2) \end{cases}$$

(2) 找出求 $X$ 和 $Y$ 的具体公式：

② 式 − 2 × ① 式： $2Y = F - 2H$ $\quad \therefore Y = \dfrac{F - 2H}{2}$

4 × ① 式 − ② 式： $2X = 4H - F$ $\quad \therefore X = \dfrac{4H - F}{2}$

以上这两步是属于数学上的方法问题，在此基础上才能编写程序（有的初学者企图在程序中直接写上方程①、②式，然后运行，以为计算机自己会解出方程，这是误解。任何的数值计算问题都首先要建立数学模型，并利用数学上的知识找出解题的方法，计算机只能进行数据的比较和运算，并输出结果）。

此题可以用 LET 语句、INPUT 语句或 READ/DATA 语句编写程序。今用 READ/DATA 语句。

```
10  READ  H, F
20  LET   COCK= (4*H-F) /2
30  LET   BABBIT= (F-2*H) /2
40  PRINT "HEAD="; H, "FEET="; F
50  PRINT "COCK="; COCK, "RABBIT="; RABBIT
60  DATA  16, 40
70  END
RUN
HEAD= 16            FEET= 40
COCK= 12            RABBIT= 4
```

读者应开始学习编写程序并上机调试自己所编的程序。

## §3.9 怎样输入和运行一个BASIC程序

在学完这一章后，已经能够编写一些简单的 BASIC 程序了，如果有条件的话，应当进行上机练习，掌握如何输入和运行一个 BASIC 程序的方法。

假设已在纸上写好了一个 BASIC 源程序，上机操作的步骤如下。

（1）开机后，先将 BASIC 解释程序调入内存，如果使用 IBM－PC（或长城 0520），可以键入“BASICA”（IBM PC 上使用的 MS BASIC 的版本），并按“回车”键，当显示屏上出现“OK”字样，表示已进入 BASIC 状态，可以输入 BASIC 的命令或语句。

（2）在每次送入一个新的程序之前，应在从键盘上打入 NEW 命令（打入 NEW 三个英文字母，再按“回车”键），以清除计算机内原来有的程序。

（3）按照写好的源程序，按行逐个打入字符。每打完一语句行都要按“回车”键，表示本行输入结束（一般 BASIC 允许一个语句行可以容纳 255 个字符，而一般显示屏上的一行只能容纳 80 个字符，如果输入的一个语句行超过 80 个字符，在输入 80 个字符后，不必按回车键，显示屏上的光标会自动移到下一行的开头，可以接着输入其余的字符）。

（4）如果发现当前正输入的语句行上（尚未按“回车”键时）打错了一个或几个字符，这时可按“←”键，每按一次“←”键，光标就回退一格，把刚才打入的字符取消（字符从屏幕消失），可以连续按若干次“←”键来删去前面几个字符，然后打入正确的字符。但如果已按了“回车”键就不能用“←”键来删字符。

（5）如果想取消程序中某一语句行，可打入该行的行号再按“回车”键，该行就从计算机内存中消除。如欲以一新的语句行代替旧的语句行，则可用原语句行的标号打入新语句行的内容。如原有语句为：

```
10  LET  A=5
```

若又打入：

```
10  LET  B=5
```

则这后一个 10 语句行就代替了前一个 10 语句行。

（6）修改完后，想检查一下全部已修改过的程序，可以打入 LIST 命令，并按“回车”键，则会在荧光屏上显示出计算机内存中的全部程序，这一工作称“列表”或“列程序清单。”

如果只想显示一部分语句，可打入：

LIST $n_1$—$n_2$

$n_1$ 和 $n_2$ 为行号。例如打入 LIST 10—40，则显示出 10 行到 40 行的全部语句行。IBM－PC、长城 0520 和 TRS－80 等计算机系统用 LIST 命令在显示屏上列表，用 LLIST 命令（LIST 前加一个 L 字母）在打印机上列表（将程序清单打印出来）。

（7）检查无误后，可打入 RUN 命令，使程序运行。如程序没有错误，则计算机就会执行程序，并按要求输出计算结果。执行完后在荧光屏上出现提示符“OK”，表示已执行完毕，可以再接受新的命令或语句。如果程序有错误，则会在显示器上显示出错信息。

（8）如果在程序运行过程中想中断运行，可按 Ctrl－C（一手按“Ctrl”键，一手按“C”键）。

（9）程序运行后不会从内存自动消失，如果打入 RUN 命令，又可重新运行一次，在打入 NEW 命令后，原有程序才消除。

(10) 键盘上有许多键上面刻着两个不同的字符。如在“3”键上，上面是一个“#”字符，下面是字符“3”。如果仅按此键则输入“3”字符，如果想输入“#”字符则应同时按“SHIFT”键和“3”键。其它键的情况类似。

(11) 如果想将显示屏上的信息全部照样在打印机上输出，可以用 Ctrl－PrtSc 键（即一手按住“Ctrl”键，另一手按下“PrtSc”键），此后凡显示屏上出现的一切字符都同时在打印机打印出来。如果再按一次 Ctrl－PrtSc，则打印机脱机（不再同步打印），信息只在显示屏上显示。如此可以反复多次使用 Ctrl－PrtSc。即：按奇数次 Ctrl－PrtSc，就挂上打印机，按偶数次则打印机脱开。（这是对长城（IBM－PC）机而言的）

应当说明，不同的 BASIC 系统的操作命令有一些细小的区别，但基本的操作步骤是相同的，遇到问题可参阅手册或请教计算机管理人员。

## 习　题

3.1　请叙述赋值语句的特点。它与数学中的等式有何不同。

3.2　如果将赋值号二侧的变量名互换，如将：

10　X＝A　　　改成：10　A＝X

会产生什么后果？

3.3　分析以下两个程序的运行结果：

程序（1）

```
10  LET  X=3
20  LET  Y=4
30  LET  X=Y
40  PRINT X, Y
50  END
```

程序（2）

```
10  LET  X=3
20  LET  Y=4
30  LET  Y=X
40  PRINT  X, Y
50  END
```

3.4　下列语句各有什么错误？

(1) LET　X＝3，Y＝4，E＝5

(2) LET　X∧2＝Y

(3) LET　X＋1＝Y－2

(4) READ　A、B、C、D

(5) DATA　2，3.8，3＋5，－6－3

(6) READ　A＋B，C

(7) INPUT　X；Y；E；

(8) PRINT　A＝SIN（3．5）

3.5　下面程序运行的结果是什么？

```
10  READ  A, B, C, D, E, N
20  LET  T=A+B+C+D+E
30  LET  MEAN=T/N
40  PRINT  "MEAN=";MEAN
50  DATA  8, 4
60  DATA  6, 15, 4, 5
99  END
```

3.6　写出下列程序的运行结果。

```
10  READ  A, B
20  DATA  1, 2, 3
25  RESTORE
```

```
30 READ C, D, E, F
40 RESTORE 50
50 DATA 4, 5, 6, 7
60 READ G, H, I, J, K, L, M, N
80 DATA 8, 9, 10, 11
90 PRINT A, B, C, D, E, F, G, H, I, J, K, L, M, N
99 END
```

3.7 写出下列程序运行后的结果。

(1)
```
10 LET X=2
20 LET Y=4
30 PRINT X, Y
40 END
```

(2)
```
10 PRINT "A", "B", "C", "D", "E"
20 PRINT 2*2, 3*3, 4*4, 5*5, 6*6, -1, 2, 4,
30 PRINT 8, -16, 0.25, 0.5, 1, 2, 4
40 END
```

(3)
```
10 PRINT 12*3; -6+3.5,
20 PRINT 1/2, 15/3, 7+8/2;
30 PRINT 2+3; 6+7, 8, 7
40 END
```

3.8 如果 A=6，B=8.5，希望计算和打印出下面的结果，请编写程序。

```
RUN
A*B=6*8.5=51
```

3.9 分别写出三个程序，以进行下列计算：

(1) $10^3-10^2$　(2) 圆半径 $r=10$，求圆面积，圆球体积　(3) $\frac{8\sin\pi}{5\times 2}$

3.10 把华氏温度 $F$ 转换为摄氏温度 $C$。公式是：

$$C=\frac{5}{9}\cdot(F-32)$$

请编程序，设 $F=64$，结果小数四舍五入。

3.11 火车作直线匀加速运动，已知初速度 $V_0=10$ 厘米/秒，加速度 $a=100$ 厘米/秒$^2$，求 $t=10$ 分钟时的速度（以千米/小时表示）。计算公式为：

$$V=V_0+at$$

3.12 计算借款额。假若你向银行借款，你估计每年能偿还款数为 $a$，拟分几年还清，年利率为 $r$，则可以借的款 $D$ 的计算公式为：

$$D=\frac{a(I^n-1)}{(I-1)\times I^n}$$

其中：$I=1+r$。假定已知：$a=240$，$n=5$，$r=0.12$，请编程序求 $D$（小数点后四舍五入）。

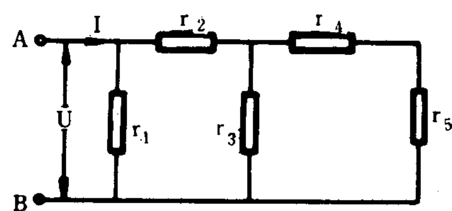



图 3. 9

3.13　有如图 3.9 所示的电路，其中 $r_1=100\Omega$，$r_2=200\Omega$，$r_3=150\Omega$，$r_4=50\Omega$，$r_5=15\Omega$，$U=100\text{V}$，求等效电阻 $r$ 和总电流 $I$。

# 第四章　选择结构程序设计

## §4.1　概　　述

在第三章中介绍了顺序程序设计。在某些情况下，为了实现特定的运行目标，只依靠顺序结构是不够的。这时根据需要流程不必完全按照语句行的行号大小顺序执行，而允许流程在程序中按一定规律转移。例如，根据某个指定的条件是否满足而决定执行 A 组语句或 B 组语句。又如，在某些条件满足时反复执行某一组语句。前者就是选择结构，后者就是循环结构。这是两种十分重要的程序基本结构。一个正式可供使用的程序很少不包含这两种基本结构的（至少其中之一）。BASIC 中有直接实现这两种基本结构的语句，也可以利用 GOTO 语句和其它语句一起来实现这两种基本结构，在介绍 IF 语句之前，先简单介绍 GOTO 语句及其使用。

## §4.2　GOTO 语句

GOTO 语句是无条件转移语句。它的一般形式为：

GOTO　**行号**

其作用是使流程无条件地转到指定的语句行去。例如：

```
10  GOTO  40
```

执行 10 语句的结果是使流程转到 40 语句行继续执行。显然 GOTO 语句改变了按行号顺序执行语句行的性质。结构化程序设计方法反对滥用 GOTO 语句，因为它可能导致程序的无规律化、非结构化（见第二章 §2. 8），产生 BS 型程序。只允许在一定条件下有限制下使用 GOTO 语句，即：为了形成选择结构或循环结构而不得不使用 GOTO 语句（见下节），只允许在一个基本结构内使用 GOTO 实现转移而不允许使用 GOTO 语句使流程从一个基本结构转到另一个基本结构中。

总之，BASIC 虽然提供了 GOTO 语句，但不提倡随便使用 GOTO 语句。滥用 GOTO 语言是造成非结构化程序的根源。只要能够不使用 GOTO 语句，就不要使用它。当然并不是禁止使用 GOTO 语句，而是必须遵循结构化原则。

## §4.3　IF 语句的概念

在 BASIC 语言中，选择结构是用 IF 语句来实现的。例如：将 a，b 两数中的大者打印出来。可以用图 4. 1 表示算法。在 BASIC 中可以用一个 IF 语句实现之：

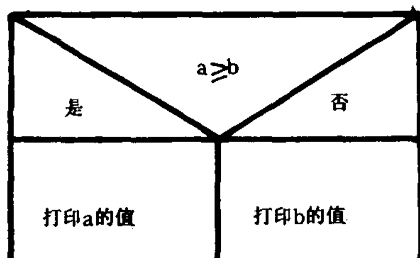


图 4.1

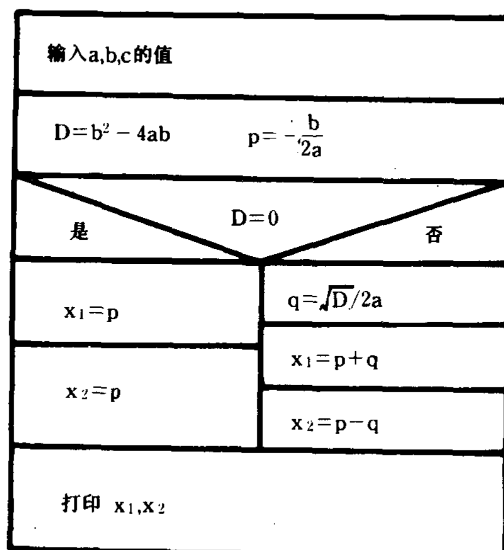


图 4.2

10 IF A>=B THEN PRINT A ELSE PRINT B

MS-BASIC 提供以下两种格式的 IF 语句：

(1) IF 〈条件〉 THEN 〈语句组〉〔ELSE 〈语句组〉〕

(2) IF 〈条件〉 GOTO 〈行号〉〔ELSE 〈语句组〉〕

方括弧内的部分表示是可选项，即可以有也可以没有。

例如以下几种 IF 语句都是合法的。

```
10  IF  X>0 THEN  PRINT X
20  IF  A>B THEN C=A : A=B : B=C
30  IF  Y>0  THEN  PRINT  Y  ELSE  PRINT -Y
40  IF  C1=C2  GOTO  100      (转到行号为 100 的语句行)
50  IF  D>E GOTO 200 ELSE PRINT D
```

说明：(1) THEN 或 ELSE 子句中的"语句组"可以包括一个或多个可执行语句。如果包括一个以上语句，则语句间用冒号分隔。

(2) 上面举例中的"IF C1=C2 GOTO 100"语句，实质上也是第一种形式的 IF 语句的变型，即："IF C1=C2 THEN GOTO 100"，而其中的"GOTO 100"是一个语句。当 THEN 后面跟 GOTO 语句时，允许省略"THEN"。

**【例4.1】** 求一元二次方程 $ax^2+bx+c=0$ 的根（只考虑实根，即 $D=b^2-4ac\geq 0$）。

根据数学知识，可先画出 N-S 流程图，即先设计出算法。见图 4. 2。

可以根据图 4. 2 所示算法编写出以下 BASIC 程序。

```
10  INPUT "please enter a, b, c"; A, B, C
20  D=B∧2-4*A*C
30  P=D/ (2*A)
40  IF D=0 THEN X1=P: X2=P ELSE Q=SQR (D )/(2*A): X1=P+Q: X2=P-Q
50  PRINT "X1="; X1,     "X2="; X2
60  END
```

```
RUN
please enter a, b, c: 1, 4, 2↙
X1=-.5858            X2=-3.4142
```

有的计算机系统用的 BASIC 不能用 ELSE 子句（例如 Applesoft BASIC）。这时可以改用其它形式的 IF 语句和 GOTO 语句来实现。

上面的程序中 40 行也可以改写为：

```
40  IF  D=0  THEN  X1=P : X2=P : GOTO  50
45  Q=SQR (D) / (2*A)
48  X1=P+Q : X2=P-Q
50  …
```

也可以用以下形式的 IF 语句实现：

```
40  IF  D=0  GOTO  48
42  Q=SQR (D) / (2*A)
44  X1=P+Q : X2=P-Q
46  GOTO  50
48  X1=P : X2=P
50  …
```

它们的流程图都可以用图 4. 2 表示。也可以画成图 4. 3 那样的传统的流程图。

有一些较早的 BASFC 版本的 IF 语句功能不够强，读者可以模仿上面的办法用系统所提供的 IF 语句来实现选择语句。可以看到：一个选择结构可以用一个 IF 语句来实现，也可以用一个 IF 语句和若干个其它语句来实现（其中可以包括一个或二个 GO-TO 语句）。显然，用 IF-THEN-ELSE 形式的语句最为方便、易读。

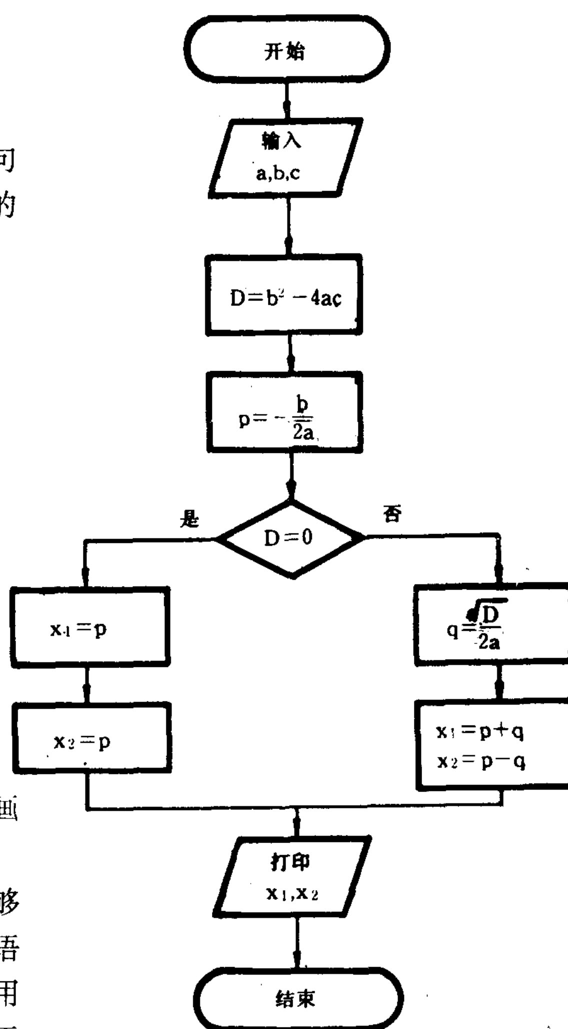



图 4.3

# §4.4 关系表达式和逻辑表达式

## 4.4.1 关系表达式和关系运算符

前面在 IF 语句中所用到的“条件”，例如“$X>0$”或“$A>B$”等，就是关系表达式。所谓关系表达式是用一个关系运算符连接两个数值量（或字符量），它的一般形式为：

**〈数值表达式〉〈关系运算符〉〈表达式〉**

BASIC 中有 6 种关系运算符：

>（大于），<（小于），=（等于），>=（大于或等于，相当于数学中的“≥”），<=（小于或等于，相当于数学中“≤”），<>（不等于，相当于数学中的“≠”）。

以下都是关系表达式合法形式：

$A > B, C = D, X >= Y, X < 0, N <> 10, X <= 8.3, A - B > X - Y, A * B > 13.6$

关系表达式有一个值，以“真”或“假”来表示。当给定的条件满足（成立）时，关系表达式的值为“真”，否则为假。例如，当 $A = 3.0, B = 5.0, X = -3.5, Y = 6.3$ 时，关系表达式的值如表 4. 1 所示：

表 4.1

| 关系表达式(即“条件”) | 条件满足否 | 关系表达式的值 |
|---|---|---|
| $A > B$ | 不满足 | 假 |
| $X >= Y$ | 不满足 | 假 |
| $X < 8.3$ | 满足 | 真 |
| $X < 0$ | 满足 | 真 |
| $A - B > X - Y$ | 不满足 | 假 |
| $A * B > 13.6$ | 满足 | 真 |

可以看出，所谓关系表达式就是将两个数值型数据进行比较，当比较的结果满足所指定的条件时，关系表达式的值就为“真”，否则为“假”。通俗地说，“真”就是给定的条件满足（或称条件成立），“假”就是给定的条件不满足（不成立）。关系运算符又称为“比较符”。对下面 IF 语句：

```
10  IF  A>B  THEN PRINT A
```

严格地说，应读为“若关系表达式“A>B”的值为真时，显示 A 的值”，为了便于理解，也可以读为“若给定条件“A>B”满足时显示 A”，或“若 A 大于 B，显示 A”。

对数值量的比较规则，按其代数值大小进行比较，例如 3 大于－3，5 不等于－5。BASIC 除了可以对数值量进行比较外，还可以对字符型数据进行比较（见本书第八章）。

在 BASIC 中没有提供逻辑型数据，因此对“真”或“假”这样的逻辑量用一个数值来代表，一般以“0”表示“假”，以一个非零的数代表“真”（在 MS－BASIC 中以“－1”代表“真”，在 Applesoft BASIC 中则以“1”表示“真”）。关系表达式的值是可以输出的，如：

```
10  PRINT  5>3, 5<3
```

将显示出：

```
-1          0
```

在 IF 语句中是根据关系表达式的值为 0 或非零来决定是否执行 THEN 子句的。假如 X=5，则

```
10  IF  X  THEN  PRINT  X
```

也能打显示 X 的值（5）。有些书上说，“当 X 不等于零时可以省写‘<>0’，”就是这个意思。但是这样写易读性差，不宜提倡。还是应该完整地写出关系表达式。

### 4.4.2 逻辑表达式和逻辑运算符

一个简单的条件可以用一个关系表达式来表示。如果要指定的是一个由几个简单的条件组成的复合条件，例如：“$X>0$ 和 $X>10$ 和 $Y>0$ 和 $Y>10$”（见图 4. 4 (a)），”$X<10$ 或 $X>20$”（见图 4. 4 (b)），就要把几个关系表达式联接组成一个逻辑表达式。例如上述的两个“复合条件”可以用下面两个式子表示：

$X>0$ AND $X<10$ AND $Y>0$ AND $Y<10$

$X<10$ OR $X>20$

这就是两个逻辑表达式。

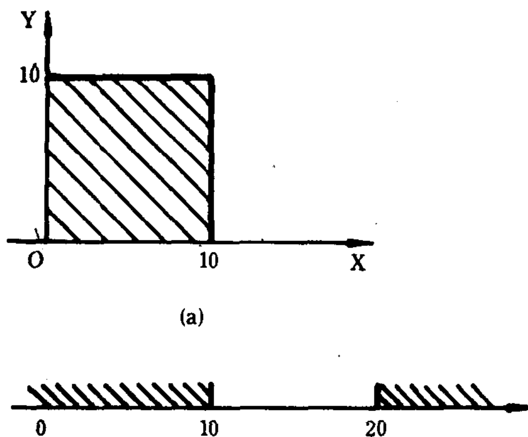



图 4. 4

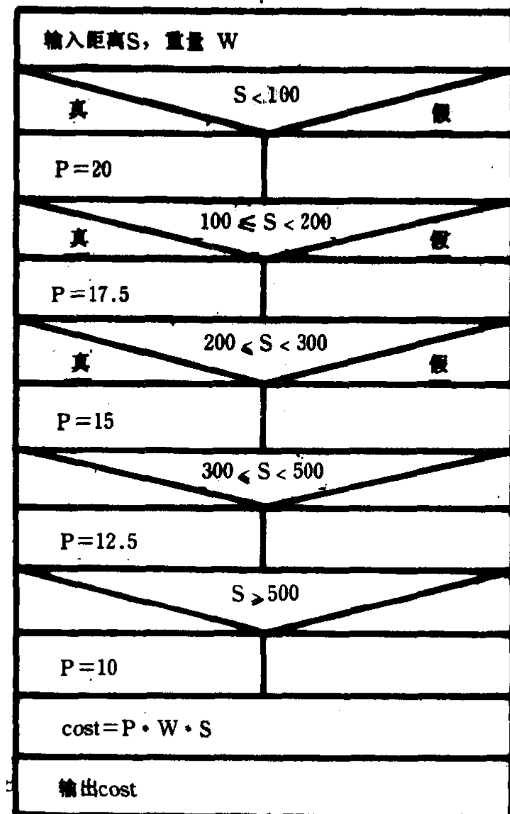



图 4. 5

BASIC 中可用的逻辑运算符有三个：

AND（与）　　如果其前后的两个简单条件都满足，则认为满足此复合条件。

OR（或）　　如果在其前后两个简单的条件之一满足，则认为满足此复合条件。

NOT（非）　　对写在其后的条件取“反”。

假设 A 和 B 代表两个关系表达式，只有当 A 和 B 都为真时，A AND B 的值才为真。只要 A 或 B 之一为真，则 A OR B 的值为真。若 A 为真，则 NOT A 的值为假，见表 4. 2：

表 4.2

| A | B | A AND B | A OR B | NOT A | NOT B |
|---|---|---|---|---|---|
| 真 | 真 | 真 | 真 | 假 | 假 |
| 真 | 假 | 假 | 真 | 假 | 真 |
| 假 | 真 | 假 | 真 | 真 | 假 |
| 假 | 假 | 假 | 假 | 真 | 真 |

如果有一个 IF 语句如下：

10 IF X>0 AND Y>0 THEN PRINT X，Y

只有当 X>0，并且 Y>0 时，才显示 X 和 Y 的值，只要“X>0”或“Y>0”两个条件之一不满足，都不会输出 X，Y 的值。又如：

20 IF X<10 OR X>20 THEN PRINT X

只要“X<10”或“X>20”二者之一满足都要执行“PRINT X”语句。譬如，$X$ 的值为－5，在执行上面的 IF 语句时，将会输出 $X$ 的值（－5）。又如：

30 IF NOT X<0 THEN PRINT $X$

其含义是：若 X 不小于 0 则显示 X 的值。假设 X 的值为 5，那么关系表达式“X<0”的值为“假”，而前面加一个逻辑运算符“NOT”表示“取反”，所以逻辑表达式“NOT X<0”的值为“真”，故应执行“PRINT X”语句。

关系表达式是最简单的一种逻辑表达式，它不包含逻辑运算符。

【例 4.2】计算运费。设货物运输价格 $P$（元）与运输距离 $s$（千米）有关。按以下公式定价（$P$ 为吨/千米运价）

$$P = \begin{cases} 20 & (\quad S < 100) \\ 17.5 & (100 \leq S < 200) \\ 15 & (200 \leq S < 300) \\ 12.5 & (300 \leq S < 500) \\ 10 & (\quad S \geq 500) \end{cases}$$

输入 $S$ 和货物吨数 $W$，要求计算出总运费。

COST（总运费）$= P * W * s$

根据题意，画出 N－S 图，判断 S 在哪一区间然后决定 $P$ 的值。见图 4.5。根据图4.5编写程序如下：

```
10  INPUT "enter distance and weight:", S, W
20  IF  S<100 THEN P=20
30  IF S>=100 AND S<200 THEN P=17.5
40  IF S>=200 AND S<300 THEN P=15
50  IF S>=300 AND S<500 THEN P=12.5
60  IF S>=500 THEN P=10
70  COST=P*S*W
80  PRINT "COST="; COST
99  END
RUN
```

```
enter distance and weight: 20, 100↙
COST=40000
```

请注意：有的读者往往容易将程序错写成以下这样：

```
   ⋮
30  IF  S<200  THEN  P=17.5
40  IF  S<300  THEN  P=15
50  IF  S<500  THEN  P=12.5
60  IF  S>500  THEN  P=10
   ⋮
```

请考虑错在何处：为什么错？试以 S=200 为例，第一个程序输出 COST=40000，第二个程序输出 COST=35000。请读者画出与第二个程序相应的 N－S 流程图。

## §4.5 IF 语句的嵌套

我们已介绍了 IF 语句可以采用以下形式：

IF〈逻辑表达式〉THEN〈语句组〉ELSE〈语句组〉

语句组中可以包括一个或多个可执行语句，如果在语句组中又包含一个 IF 语句，就形成了 IF 语句的嵌套。例如：

```
10  IF  X>Y  THEN IF X>Z THEN PRINT "MAX="; X
```

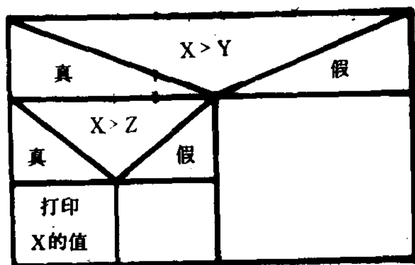



图 4.6 (a)

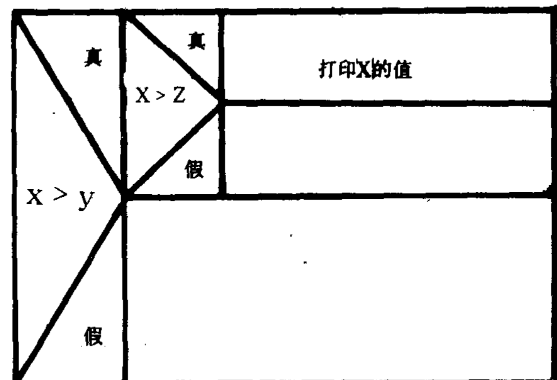



图 4.6 (b)

其作用是：先求关系表达式“$X>Y$”的值，如果为真（即 $X$ 的值大于 $Y$ 的值），然后再求“$X>Z$”的值，如果也为真，才执行 THEN 后面的 PRINT 语句。流程图见图 4. 6 (a)，请注意，内层的 IF 语句是嵌套在外层 IF 语句中的 THEN 子句中的。如果 X≯Y，则不会执行内层的 IF 语句。

当嵌套层次多时，画 N－S 流程图时会出现每个“真”和“假”的框分得愈来愈小，有可能写不下相应的操作内容。为此，作者建议可以将选择结构横过来画，如图 4. 6 (b) 所示形式。这种画法把嵌套 IF 的各个“真”、“假”操作框向纵向展开，而纵向的篇幅比横向的篇幅大得多，从而较好地解决了上面提出的问题。在看图时依然从上到下看下来，把一个 IF 语句

所对应的选择结构看作一个整体。

【例 4.3】 输入 $A$ 和 $B$ 的值，要求显示出其中值较小的数。

```
10  INPUT "A, B=", A, B
20  IF  A<=B THEN IF A<B THEN PRINT A; ELSE PRINT "NEITHER "; ELSE PRINT B;
30  PRINT "IS  SMALLER"
40  END
```

相应的 N-S 流程图见图 4. 7 (a) 和 (b)

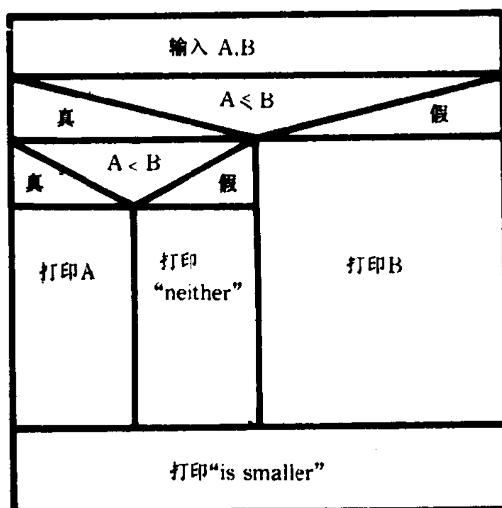



图 4.7 (a)

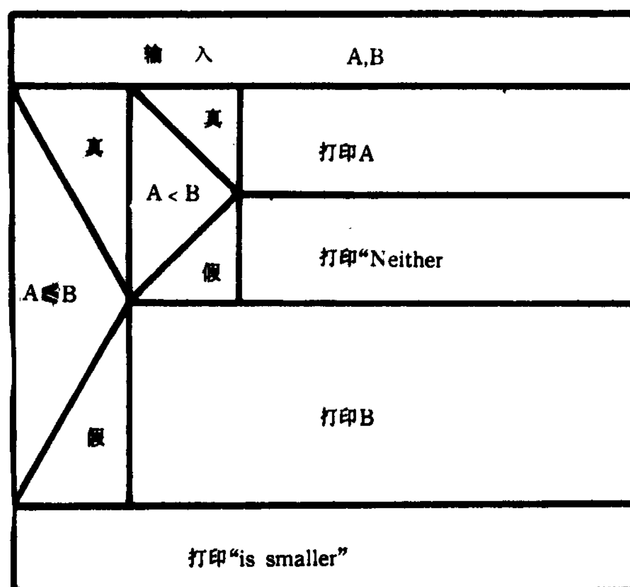



图 4.7 (b)

它是这样执行的：先检查 $A \leq B$？如果不满足 A≤B，则跳过 THEN 后面的部分，直接去执行最后一个 ELSE 后面的语句 (PRINT B)。如果 A≤B，则执行 THEN 后面部分，此时再检查 A<B? 如果满足，则打印 A，如不满足（即 A=B），则打印 "NEITHER"。然后执行 30 语句。对照流程图看，其作用是很清楚的。

运行开始后，输入二个数据给 A 和 B，程序检查比较 A 和 B 的值，将值小的打印出来。当 A 小时，打印："A IS SMALLER"。当 A=B 时，打印："NEITHER IS SMALLER"。

```
RUN
A, B=3,6↙
 3 IS SMALLER
```

在程序的 20 行是一个嵌套的 IF 语句，包含有两个 "IF"、两个 "THEN"，两个 "ELSE"。要弄清楚哪个 ELSE 与哪个 IF-THEN 配对。配对的原则是：先从最内层的 ELSE 开始找，ELSE 总是与离它最近的且在它前面的 IF 配对，如此逐层配对。把它写成下面锯齿形状比较容易看出各层的嵌套关系。

```
   ┌IF  A<=B  THEN
10 │┌IF  A<B  THEN PRINT A;
   │└ELSE PRINT "NEITHER";
   └ELSE PRINT B;
```

BASIC 的一个 IF 语句必须在一个语句行（最多可包含 255 个字符）写完，这样的写法清晰度提高了，但相当于在语句行中增加了许多空格，增加了语句的长度。当长度超过 255 时就会出错。所以尽量不要使用多层的 IF 嵌套，否则很容易弄错嵌套的关系。本例也可以用以下程序实现：

```
10 INPUT "A, B=", A, B
20 IF A>B THEN PRINT B;
25 IF A<B THEN PRINT A;
28 IF A=B THEN PRINT "NEITHER";
30 PRINT "IS SMALLER"
40 END
```

或者：

```
10 INPUT "A, B=", A, B
20 IF A>B THEN PRINT B;: GOTO 30
25 IF A < B THEN PRINT A; ELSE PRINT "NEITHER";
30 PRINT "IS SMALLER"
40 END
```

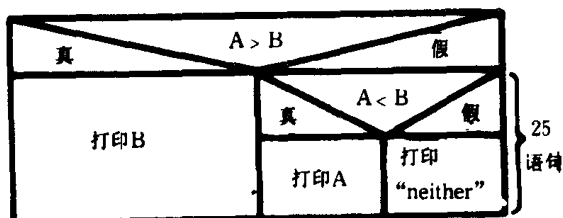



图 4.8

在这个程序中，用两个语句行（20 和 25 行）来实现图 4. 8 所示的选择结构的嵌套。注意，只有在 $A \not> B$ 的条件下才会执行 25 行。25 行的作用是实现 20 语句中"A>B"为假时应进行的操作，相当于 20 语句中有一个 ELSE 部分。

**【例4.4】** 有一函数：

$$y=\begin{cases}+1 & (x>0)\\ 0 & (x=0)\\ -1 & (x<0)\end{cases}$$

输入 $x$，要求打印出 $y$ 的值。

可编出程序如下：

```
10 INPUT "X=", X
20 IF X>0 THEN Y=1 ELSE IF X=0 THEN Y=0 ELSE Y=-1
30 PRINT "X="; X, "Y=" Y
40 END
```

根据前面叙述的嵌套原则，第一个 ELSE 与第一个 IF 配对，第二个 ELSE 与第二个 IF 配对。流程图见图 4. 9。内层的 IF 语句是嵌套在外层 IF 语句的 ELSE 子句中的。

也可以将 20 语句改写为：

```
20 IF  X>=0  THEN IF X>0 THEN Y=1 ELSE Y=0  ELSE Y=0
```

此时，内层的 IF 语句是嵌套在外层 IF 语句的 THEN 子句中的，流程图见图 4. 10，请注意 ELSE 与 IF 的配对关系。有人写出以下程序：

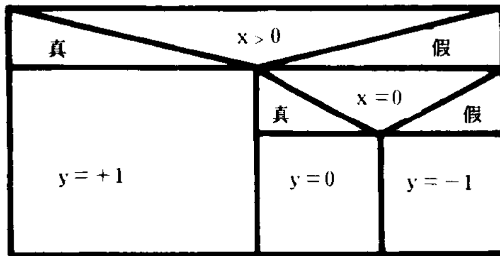



图 4.9

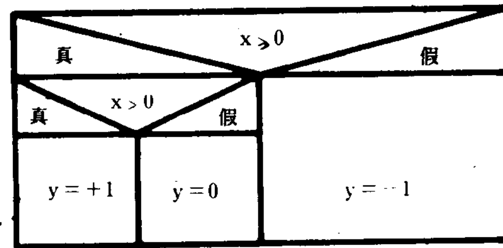



图 4.10

```
10  INPUT "X="; X
15  Y=0
20  IF X>=0 THEN IF X>0 THEN Y=1 ELSE Y=−1
30  PRINT "X="; X, "Y="; Y
40  END
```

编程序者的意图是实现图 4. 11 的流程。但由于现在只有一个 ELSE，它与哪一个 IF 配对呢？根据规定，它与第二个 IF 配对而不与第一个 IF 配对，因此它对应的不是图 4. 11，而是图 4. 12，这显然不符合题目要求。当 X=0 时，X=−1 而不等于 0，而当 X<0 时，Y 却等于 0。

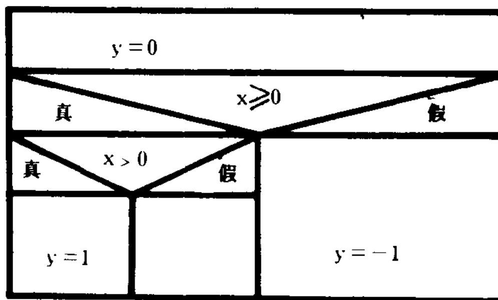



图 4.11

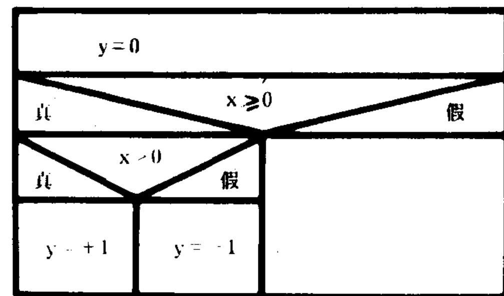



图 4.12

因此，当使用嵌套关系时一定要十分仔细，防止出现逻辑错误。

**【例 4.5】** 求一元二次方程 $Ax^2 + Bx + C = 0$ 的根。考虑 D=B²−4AC 大于 0，等于 0 和小于 0 三种情况。

第二章图 2. 17 已给出流程图。

用嵌套的 IF 语句编写出程序：

```
10  INPUT "A, B, C="; A, B, C
20  LET D=B*B−4*A*C
30  IF D<0 THEN PRINT "COMPLEX ROOTS" ELSE IF D>0  THEN X1= (−B+SQR (D) )/(2*A):
    X2= (−B−SQR (D) )/(2*A): PRINT X1, X2 ELSE X=−B/ (2*A): PRINT X
40  END
```

请读者画出与此程序相应的流程图（与图 2. 16 略有差别）。30 语句中内层的 IF 语句是

嵌套在外层 IF 语句的 ELSE 子句中的。当然也可以将内层 IF 嵌套在外层 IF 语句的 THEN 子句中。但是，最好把内层 IF 嵌套在外层 IF 的 ELSE 部分（尤其是当内层的 IF 只有 THEN 部分而没有 ELSE 部分时），这样逻辑关系不易搞错。请看下面示意：

IF…THEN…ELSE…IF…THEN…ELSE…

内层 IF 与外层 IF 之间有外层的 ELSE 隔开，不致混淆。如果把它嵌套在外层 IF 的 THEN 部分：

IF…THEN…IF…THEN…ELSE…ELSE…

内层

外层

当内层的 IF 只有 THEN 部分而没有 ELSE 部分时，就成了：

IF…THEN…IF…THEN…　ELSE

程序设计者的原意是将有下划线的 IF－THEN 部分作为内层 IF 语句，但由于 ELSE 语句与其最近的 IF 配对，因此 ELSE 语句成为内层 IF 语句的一部分（如虚线所示），这就出现逻辑错误。

应该说，由于 BASIC 规定只能在一个语句行中写出整个 IF 语句（包括嵌套部分），而不能分行写（如 FORTRAN 的块 IF 或 PASCAL 的 IF 语句可以分行写），使 IF 的嵌套使用和阅读都不太方便。可以改用几个语句来实现选择结构的嵌套。请读者参考例 4．3 的方法改写例 4．5的程序。

## §4.6　多分支选择语句（ON－GOTO 语句）

IF 语句可以从两条路径中选择其一。但有时遇到的问题是要求从多条路径中选择其一，例如，根据学生考分将成绩分段（90 分以上，80～89 分，70～79 分，60～69 分，60 分以下），分别进行处理，又如银行存款有不同的种类，根据不同类别决定不同的利息。当然可以用嵌套的 IF 语句来实现多分支选择，但 IF 语句会比较长，程序清晰度降低，BASIC 提供 ON－GOTO语句实现多分支选择。ON－GOTO 语句的一般形式为：

ON〈**算术表达式**〉GOTO〈**行号** 1〉，〈**行号** 2〉，…〈**行号** n〉

它的作用是：先计算算术表达式的值，若值为小数则按四舍五入原则处理得一整数。如果此整数值为 i 则流程转到行号 i 指出的语句行去继续执行。表达式的值应大于或等于 1，小于或等于 n，n 最大值为 255，即最多可以有 255 个分支。如果表达式的值小于 1 或大于 n，则执行 ON－GOTO 语句的下一语句（即不执行本 ON－GOTO 语句）。如果有以下语句：

```
30  ON  X  GOTO  100, 300, 200, 500
```

当 X＝1 时，流程转到 100 行，X＝2 时，流程转到 300 行，X＝3 时，流程转到 200 行，X＝4 时，流程转到 500 行。

ON－GOTO 语句相当于一个“多路开关”，当 X 为某一个值时接通某一通道。见图 4．13。

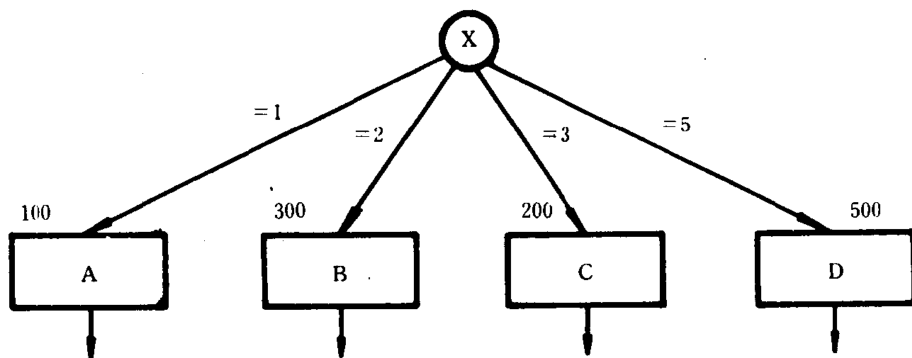



图 4.13

**【例 4.6】** 售货价格随购货数量而异，买 100 个以上（含 100，下同）的为 95 折，200 个以上的 90 折，300 个以上 85 折，400 个以上的 80 折，500 个以上 75 折。设商品价为 65 元。要求：在输入购买商品个数后，打印出总货款。

编程序思路：题目规定购货每增加 100 个折扣就换一档，假设购商品数 $Q$，则 INT（$Q$/100）的值为：

$$INT(Q/100)=\begin{cases}0 & (\qquad Q<100)\\1 & (100\leq Q<200)\\2 & (200\leq Q<300)\\3 & (300\leq Q<400)\\4 & (400\leq Q<500)\\5 & (500\leq Q<600)\end{cases}$$

$Q$ 若大于 500，一律按 75%。今用 ON－GOTO 语句来处理。由于 ON－GOTO 语句中的表达式（V）值必须大于 0（最小为 1），故程序作以下处理：

```
10  P=65       (商品价)
20  INPUT "Q=", Q (输入购买款)
30  V=INT (Q/100) +1
35  IF V>6 THEN V=6
40  ON V GOTO 100, 200, 300, 400, 500, 600
100  R=1: GOTO 1000 (R 为折扣)
200  R=0.95: GOTO 1000
300  R=0.9: GOTO 1000
400  R=0.85: GOTO 1000
500  R=0.8: GOTO 1000
600  R=0.75
1000  T=P*Q*R      (计算总货款)
1010  PRINT "AMOUNT="; T
1010  END
RUN
```

Q=250↙

AMOUT=14625

程序中 100～500 行中都有一个 GOTO 语句以便使它们汇合到同一个出口点（见图 4. 14)，这才符合结构化原则（请思考，如果没有这些 GOTO 语句会出现什么结果)。此程序是结构化的，可以画出 N－S 图（见图 4. 15)，请注意，多分支选择在 N－S 图中的表示形式。

请读者注意分析 30 语句，为什么要加 1?因为使用 ON－GOTO 语句必须使 $V$ 的最小值为 1。35 语句使 $Q$ 大于 500 时，V 的值都为 6（都按 75 折处理)。40 语句根据 V 的值分别转去 100，200，300，400，500，600 行，相应地使折扣率 R 分别等于 1，0.95，0.9，0.85，0.8，0.75。注意，每种情况处理后都应汇集到 100 语句行，计算总数额。

如果多分支选择中，分支数很多，而且相应的操作内容比较复杂，有可能在窄小的框内写不下，可以将多分支选择结构改画成图 4. 16形式。

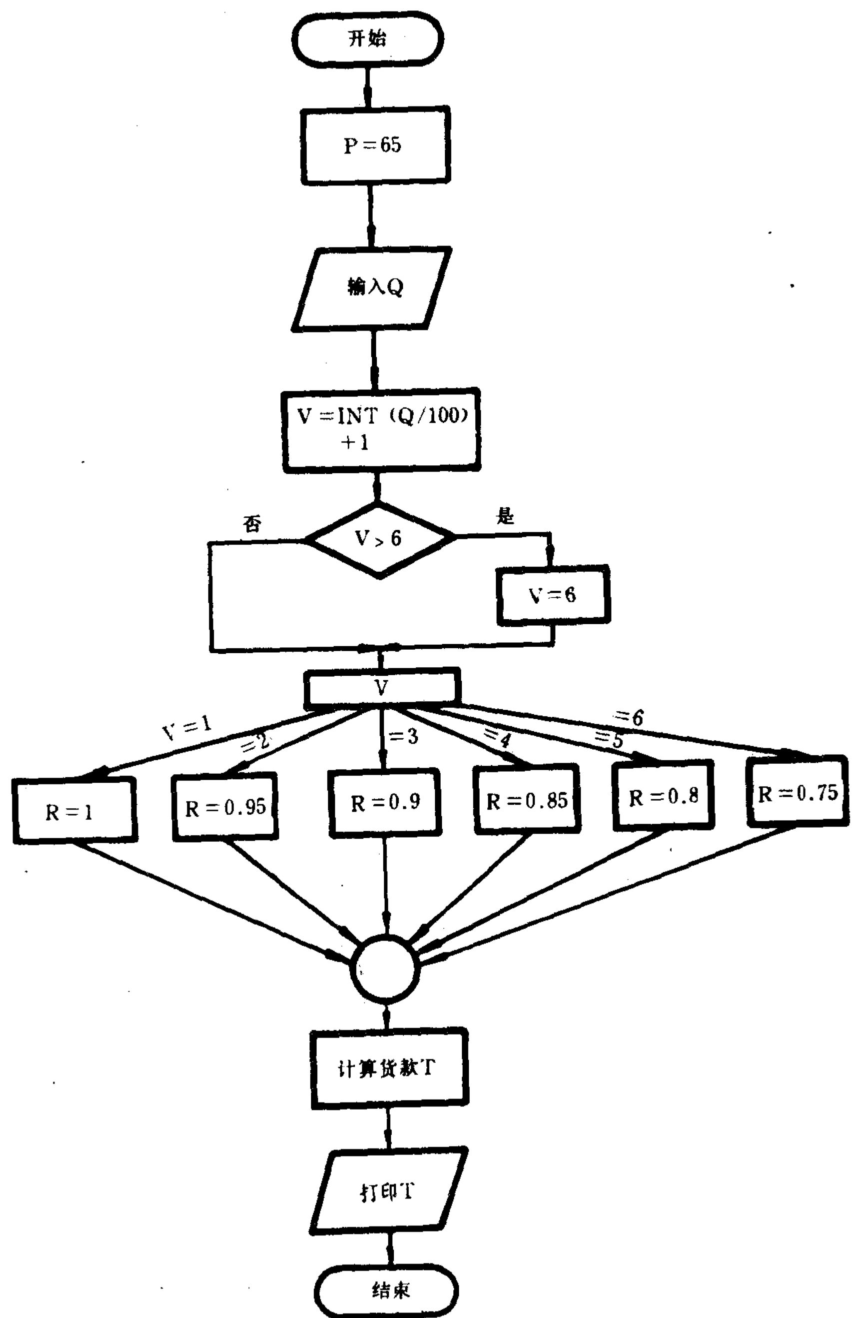



图 4. 14

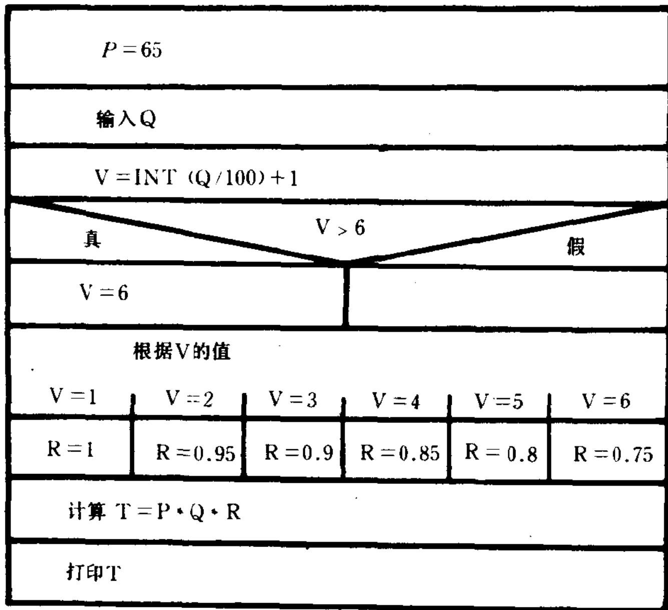



图 4.15

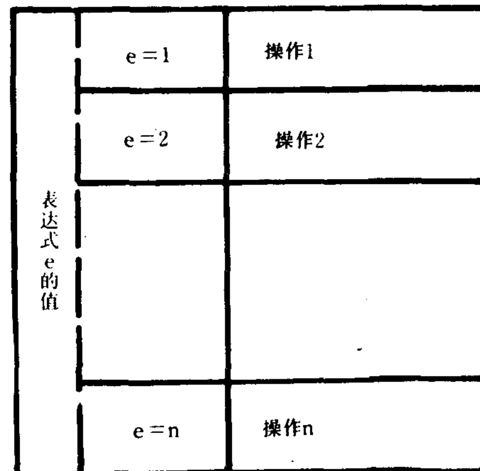



图 4.16

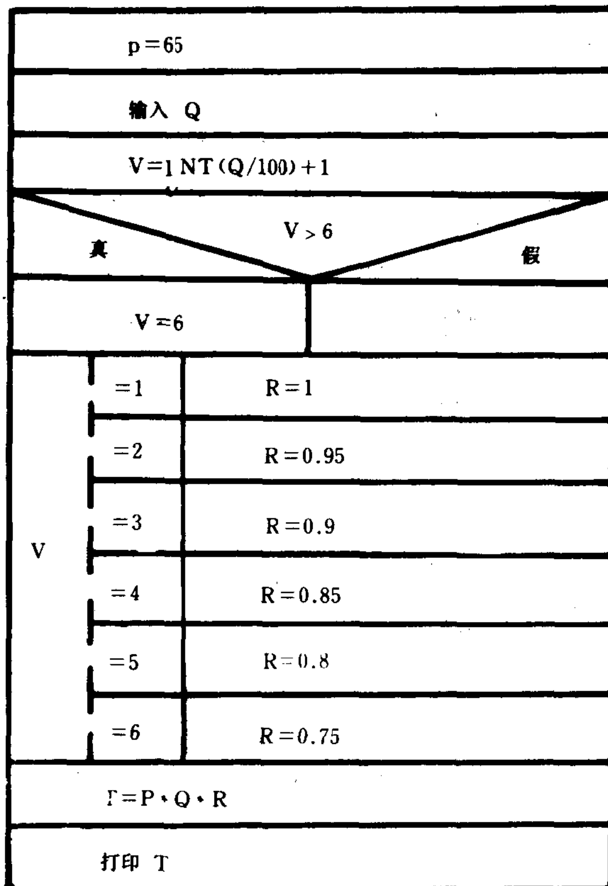



图 4.17

例如图 4.15 可画成图 4.17 形式。究竟用图 4.15 或图 4.17 形式由读者根据情况自定。二者都同样是方便易读的。

由于用 ON-GOTO 语句实现多分支选择结构要用多个 GOTO 语句使出口汇合在一起，这样虽然符合结构化原则，但并不太理想，易读性差些。在学到子程序一章（第七章）时，可以用ON-GOSUB 语句代替 ON-GOTO 语句，那样更好些。

# § 4.7 程 序 举 例

**【例4.6】** 给出三角形的三边 $a$、$b$、$c$，要求计算出三角形的面积。

首先要验证 $a,b,c$ 三条线段能否构成一个三角形。构成三角形的必要而充分的条件是：二边之和大于第三边。即：$a+b>c$，$b+c>a$，$c+a>b$。

三角形的面积 area= $\sqrt{s\cdot(s-a)\cdot(s-b)\cdot(s-c)}$

其中，$s=\dfrac{a+b+c}{2}$

可以编写程序如下：

```
10  INPUT "enter a, b, c:", A, B, C
12  S= (A+B+C) /2
20  IF (A+B>C) AND (B+C>A) AND (C+A>B) THEN PRINT "area="; SQR (S* (S—A) *
      (S—B) * (S—C)) ELSE PRINT "data error!"
30  END
```

运行情况如下：

enter a, b, c: 4, 3, 5↙

area=6

**【例 4.7】** 给出三个数 $a,b,c$，要求按大小顺序将它们打印出来。

先进行算法设计。可以用自顶向下、逐步细化的方法进行。先进行顶层设计。见图 4. 18。

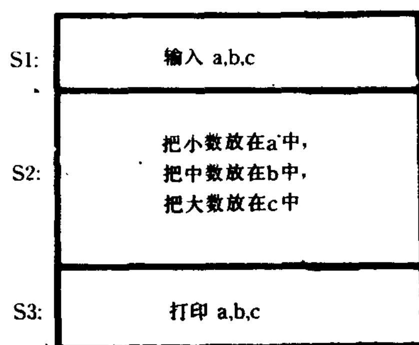



图 4.18

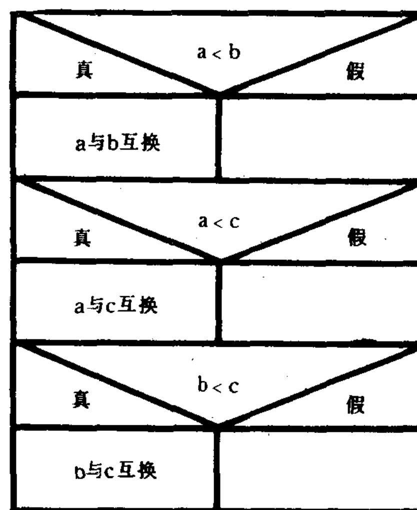



图 4.19

一般习惯以 S（英文 Step（步）的第一个字母）代表"步骤"，S1，S2，S3 分别代表"第一步"、"第二步"、"第三步"。其中 S1，S3 这两步已足够简单，无法再细化了。S2 还不够具体。怎样才能实现 S2 的要求呢？要进一步具体化。对 S2 可以细化如下：

S 2.1 将 $a$ 和 $b$ 比较，将大者放在 $a$ 中，小者放在 $b$ 中。

S 2.2　将 $a$ 和 $c$ 比较，将大者放在 $a$ 中，小者放在 $c$ 中。（此时 $a$ 已是三者中最大的数了）。

S 2.3　将 $b$ 和 $c$ 比较，将大者放在 $b$ 中，小者放在 $c$ 中。（此时 $b$ 为次大的数，$c$ 为最小数）

用N－S流程图表示S2步骤。（图4．19）。至此S2．1，S2．2，S2．3已不必（也不能）再细化了，它们可以用三个IF语句来实现。根据流程图可以编写出程序如下：

```
10  INPUT "A, B, C=", A, B, C
20  IF A<B THEN T=A: A=B: B=T
30  IF A<C THEN T=A: A=C: C=T
40  IF B<C THEN T=B: B=C: C=T
50  PRINT A; B; C
99  END
RUN
A, B, C=3, 2, 5↙
5  3  2
A, B, C=5, 4, 7↙
7  5  4
```

**【例4.8】**　整存零取的计算

（1）给定每次支款的期限、款额、全部取完的年数、以及年利率。可以计算出开始时一次应存入的款数。例如，准备在9年内每隔半年支取90元（在每半年的最后），年利率为5%，问现在一次应存入多少钱?

（2）给定的项目同上，但支取时间改为每一段期限的开始，例如在每半年的开始时去取。

（3）给定开始时一次存入的款数、全部支取完的时间（年）、多少时间取一次、以及年利率。需要求：每次可取多少钱。例如，一次整存5000元，年利率8%，准备5年内取完，每半年取一次（在第6个月的最后取），问每次可以取多少钱：

以上均按复利计算，每取一次计算一次。

先找到处理这类问题的计算公式：

复利率　$I = \dfrac{\text{年利率}}{\text{每年支取的次数}}$

应整存的款项（如果“期末”取）＝准备每次支取数 $\times Y \div I$

应整存的款项（如果“期初”取）＝准备每次支取数 $\times Y \div I \times (I+1)$

整存后每期可支取数＝整存款数 $\times I \div Y$

其中：$Y = 1 - (I+1)^{-n}$（n为支取的分期数，其值等于支取完的年数×每年支取次数）

设以D代表存取方式，当输入D＝1时，表示按题目中（1）方式（期末取），如输入 $D=2$，则按（2）方式（期初取）。以 $A$ 代表每期支取款数（对（1）（2）方式）或整存款数（对（3）方式）。$R$ 代表年利率。$L$ 代表一年取几次。$n$ 代表几年取完。

画出N－S图，见图4.20。　编写程序如下：

```
10  INPUT "MODE"; D        （输入存取方式）
20  INPUT "AMOUNT"; A      （每期支取数或整存款数）
30  INPUT "RATE"; R        （年利率）
```

```
40  INPUT "NUMBER OF INSTALLMENTS"; L
                              (每年取几次)
50  INPUT "TERM"; N       (几年取完)
110  I=R/L                (每期的复利)
120  N=N*L                  (分期数)
130  Y=1- (I+1) ∧ (-N)
140  IF  D=2  THEN  200
150     M=INT (A*Y/I+0.5) (规定"期末"取
    时应整存的款)
160     PRINT "TERM-END CRNT.  PR.", M
170     S=INT (A*Y/I* (I+1) +0.5) (规定
    为"期初"取时应整存的款数)
180     PRINT "TERM-BEGINNING CRNT.  PR."
     , S
190  GOTO 220
200     E=INT (A*I/Y+0.5)
210     PRINT "OUTSTANDING AMOUNT AT TERM END", E
220  REM
999  END
```

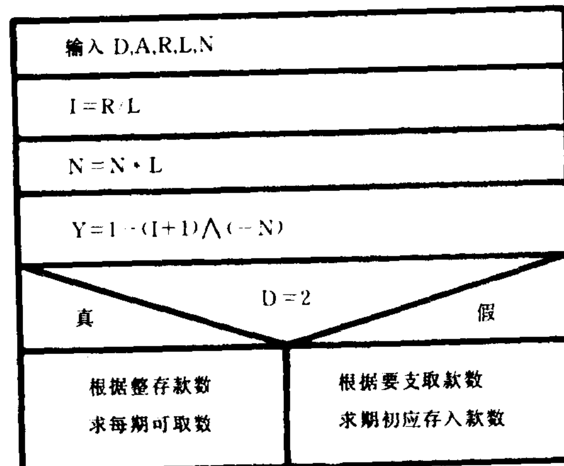



图 4.20

在程序开始运行时，先由键盘输入"存取方式"。当输入"1"时，表示已知零取款数和期限、利率、希望求整存款数（即本题开始时列出的（1）、（2）两项）。当输入"2"时，表示已知整存款数数和支取期限和利率，希望求每次可支取多少钱（即本题的第（3）项）。

运行情况为

```
RUN
MODE? 1↙
AMOUNT? 90↙        (每次取 90 元)
RATE? 0.05↙        (年利率 5%)
NUMBER OF INSTALLMENTS? 2↙        (每年取二次)
TERM? 9↙        (9 年取完)
TERM__END CRNT.  PR.            1292     (若"期末"取，应一次存入 1290 元)
TERM__BEGINNING CRNT.  PR.      1324     (若"期初"取，应一次存入 1324 元)
```

另一次运行记录

```
RUN
MODE? 2↙
AMOUNT? 500↙        (一次存入 5000 元)
RATE? 0.08↙        (年利率 8%)
NUMBER OF INSTALLMENTS? 2↙     (一年取二次)
TERM? 5↙        (五年取完)
OUTSTANDING AMOUNT AT TERM END     616 (每次在"期末"时可取 616 元)
```

请注意，程序中 140～220 语句构成一个选择结构，220 语句只是作为"汇合点"之用。

程序中 170 和 200 语句中括弧内加0.5的作用是"四舍五入"，目的是为了得到整数（元）而不要求输出角和分。在此顺便介绍一些有关四舍五入和保留指定小数位数的方法。

（1）如果要求对个位数 X 进行四舍五入处理，可以按下式进行：

INT （X+0.5）

例如，$X=39.67$，$X+0.5=40.17$，再取整得 40。若 $X=39.47$，则 $X+0.5=39.97$，取整得 39。

（2）如果想保留小数点后一位数字。可以：

INT （X * 10） /10

例如，X=39.67，X * 10=396.7，INT （396.7） =396，再除以 10，得39.6。保留了一位小数。有人会奇怪乘以 10 又除以 10，不等于不乘不除吗？请注意：$X$ 先乘 10，再取整，最后再除以 10。乘 10 的作用是使 X 的小数点右移一位（39.67变为396.7），取整的作用是将 X 的第二位小数去掉，再除 10 为了将小数点左移一位（到原位置）。用这个办法去掉第二位小数。同理，如果想保留二位小数，可以用下式：

INT （X * 100） /100

想保留 n 位小数，用：

INT （X * 10∧N） /10∧N

（3）如果想保留一位小数，对第二位小数进行四舍五入处理，可以用下式：

INT （X * 10+0.5） /10

例如，$X=39.67$，乘以 10 得396.7，加0.5得397.2，取整得 397，再除以 10，得39.7，符合题意。

如果要保留 $n$ 位小数，对第 $n+1$ 位小数四舍五入，该如何处理？

**【例 4.9】** 有一函数

$$y=\begin{cases}10+x20 & (当\ 0\leq x<10)\\ 20-(x-30) & (当\ 10\leq x<30)\\ 20 & (当\ 30\leq x<40)\\ 10-(x-50) & (当\ 40\leq x<50)\\ & (当\ 50\leq x<60)\end{cases}$$

见图4.21。

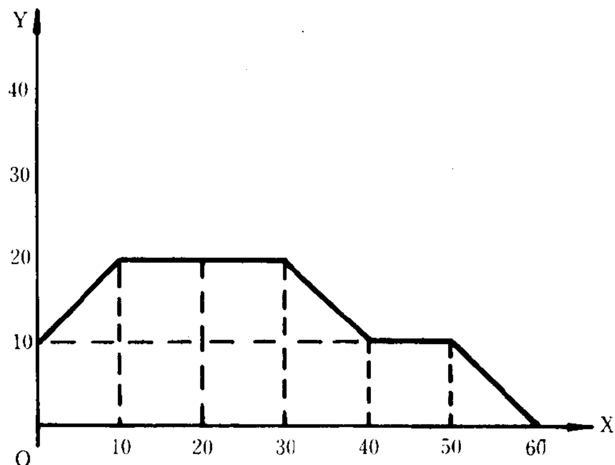


图 4.21

从图可以看出每一段直线所对应的 $X$ 值的范围都是 10 的倍数，可以将宽度 10 作为一个区间单位，第一个区间为〔0，10），第二个区间是〔10，20），第三个区间是〔20，30），第四个区间是〔30，40），第五个区间是〔40，50），第六个区间是〔50，60）。$x$ 由 0 开始变化其值，每变到 10 的倍数时，就改变 $y$ 的计算公式。由于这是根据 x 的值而选择不同的计算 y 公式，因此是多分支选择，可以用 ON－GOTO 语句。而 ON－GOTO 语句中的算术表达式的值每增加值 1 就会改变转向的语句（见§4. 6），所以我们必须找到这样的规律：使 x 每变化 10 就使“算术表达式”的值增加 1。可以找到这样的关系：

N=INT （X/10） +1

例如 0≤X<10 时，N=1，10≤X<20 时，N=2，20≤X<30 时，N=3，…。可写出下面程序：

```
10  INPUT "X=", X
20  IF X<0 OR X>60 THEN 10
30  N=INT (X/10) +1
40  ON N GOTO 50, 60, 60, 70, 80, 90
50  Y=10+X: GOTO 100
60  Y=20: GOTO 100
70  Y=20- (X-30): GOTO 100
80  Y=20: GOTO 100
90  Y=10- (X-50)
100  PRINT "X="; X, "Y="; Y
999 END
```

运行情况如下：

① X=5↙

X=5　　　　Y=15

② X=15↙

X=15　　　　Y=20

请分析：为什么在 ON - GOTO 语句中有两个相同的行号（60)？

## 习　　题

4.1　编程序，求 $y$：

$$y=\begin{cases}x^2+x+1 & (x\leq 0)\\ x^2-x+1 & (x<0)\end{cases}$$

4.2　铁路托运行李，从甲地到乙地，按规定每张客票托运行李不超过 50 公斤时，每公斤0.13元，如超过 50 公斤，超过的部分按每公斤0.20元计算。如果行李重量为 $W$ 公斤，运费为 $X$ 元。计算公式为：

$$x=\begin{cases}0.13\times W & (W\leq 50)\\ 50\times 0.13+(W-50)\times 0.20 & (W>50)\end{cases}$$

4.3　输入 $a,b,c$ 三个数，要求将绝对值最大者打印出来。

4.4　给定一个年份，判断它是否闰年。闰年的条件是：能被 4 整除但不能被 100 整除，或者能被 100 整除且能被 400 整除。(例如 1989 不是闰年，1992，2000 年是闰年)。

4.5　给定一个整数，判断它能否被 3，5，7 整除，并根据判断结果打印出以下信息之一：①能同时被 3，5，7 整除；②能被其中两数（说明哪两数，如 3，5；或 5，7；或 3，7）整除；③只能被其中一个数整除（说明是哪一个数）整除。④不能被 3，5，7 整除。

4.6 某班期考三门课，其中两门为主课，要求挑选出符合下列条件之一的学生：①三门课总分高于 270 分者；②两门主课均在 95 分以上者；③一门主课为 100 分，其它两门在 80 分以上者。输入学生学号和三门课成绩（主课 C1，C2，非主课 C3），如符合以上条件者，打印出学号和三门课成绩以及总分和平均成绩。

4.7 设一函数 $x(t)$，见图4.22，即：

$$x=\begin{cases}\text{无定义} & (t<0)\\ x_0+\dfrac{(a-x_0)t}{t_1} & (0\le t<t_1)\\ a & (t_1\le t<t_2)\\ \dfrac{a(t_3-t)}{t_3-t_2} & (t_2\le t<t_3)\\ 0 & (t_3\le t)\end{cases}$$

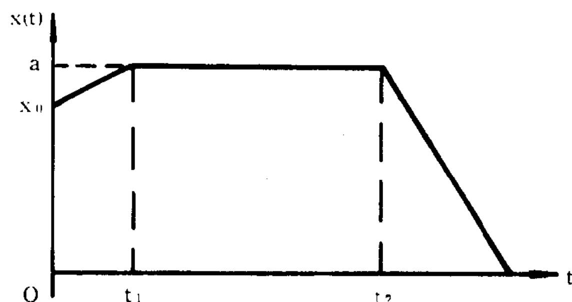


图 4.22

编程序，先输入 $a, x_0, t_1, t_2, t_3$ 再输入 $t$，然后程序输出 $x$ 的值。

4.8 有一圆，圆心坐标为 (0，0)，半径 $r=10$。见图 4. 23。编程序，输入任一个点 A 的坐标 $x, y$，判断该点在圆内，或圆周上，或圆外，打印出相应的信息。

4.9 计算税金。按不同收入（$A$）纳税。办法如下：

| | |
|---|---|
| $A\le500$ | 不交税 |
| $500<A\le1000$ | 超过 500 元的部分，交税 20% |
| $1000<A\le1500$ | 除 1000 元以内的按以上办法计算外，超过 1000 的部分按 25%计算 |
| $1500<A\le2000$ | 除 1500 元以内的按以上办法计算外，超过 1500 的部分按 30%计算 |
| $2000<A\le2500$ | 除 2000 元以内的按以上办法计算外，超过 2000 的部分按 35%计算 |
| $2500<A\le3000$ | 除 2500 元以内的按以上办法计算外，超过 2500 的部分按 40%计算 |
| $3000<A$ | 除 3000 元以内的按以上办法计算外，超过 3000 的部分按 50%计算 |

编程序，输入收入 A，输出应交税款 Tax。

4. 10 有一城市，周围有四个卫星城，土地的价格分别为：中心城内每亩 20000 元，卫星城内每亩 10000 元，其它地方每亩 5000 元。设中心城和卫星城均为圆形。圆心的坐标分别为 O (0，0)，$O_1$ (50，50)，$O_2$ (−50，50)，$O_3$ (−50，−50)，$O_4$ (50，−50)。中心城的半径为 18 千米，卫星城的半径为 10 千米（图 4. 24)。编程序，输入任一点坐标（$x, y$）和欲买地亩数，要求输出①该点坐标；②是否在中心城内或卫星城内；③每亩地价以及总购地所需款额。

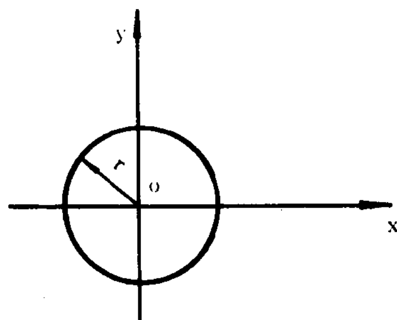



图 4.23

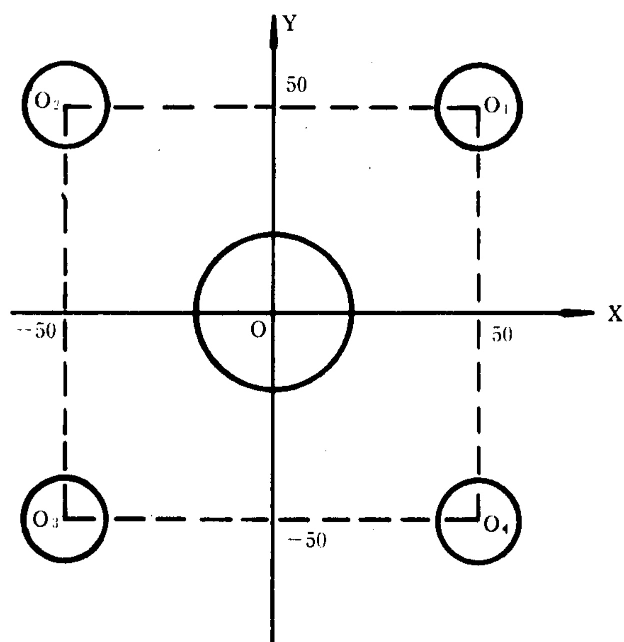



图 4.24

# 第五章　循环结构程序设计

## §5.1　概　　述

在所处理的问题中，往往遇到一些需要循环处理的问题，例如需要对一个班的 30 名学生的成绩进行检查，将不及格者打印出来；对 1～100 间各数逐个检查它们是否素数；对若干个数进行累加（例如求 $\sum_{n=1}^{100} n$）；…等。解决这些问题所用的算法，都要用到循环结构。在第二章§2.8 中已介绍了循环结构的概念。在 BASIC 中可以利用不同的方法来实现循环结构，例如，可以用 GOTO 语句形成循环。

**【例 5.1】**　累加器程序。将键盘每次输入的数据进行累加（例如用于运动会的计分，或商店、银行的算账）。这个程序使计算机用起来如同一个计算器。

```
10 LET T=0
20 INPUT "X="; X
30 LET T=T+X
40 PRINT "T="; T
50 GOTO 20
99 END
RUN
X=? 8↙
T=8
X=? ↙ 13
T=21
X=? -12↙
T=9
X=? 11.52↙
T=20.25
X=?
```

当使用者从键盘打入一个数据后，计算机马上打出当前的累计值。然后 GOTO 语句使流程转回 20 语句要求再输入一个数。每输入一个 X 值后，就累计出总和。这是一个无终止的循环，程序不会自动终止。如果不想继续输入数据，必须用人工干预强制使程序终止运行（按"Ctrl"＋"C"键或"Ctrl"＋"Break"键）。请考虑如果将"GOTO 20"改为"GOTO 10"，会出现什么结果？

**【例 5.2】**　输入华氏温度 $F$，要求转化为摄氏温度 $C$ 和绝对温度 $Z$。公式是：

$$C=\frac{5}{9}(F-32)$$

$$Z=273.16+C$$

可以编写出以下程序：

```
10 PRINT " F", " C", " Z"
20 READ  F
30 C=5/9* (F-32)
40 Z=273.16+C
50 PRINT  F, C, Z
60 GOTO 20
70 DATA 32, 135.5, 68, 3, 12.7, 87.4, 99
100 END
```

程序运行时，从 DATA 语句中读第一个数据 32⇒ $F$，然后计算 $C$ 和 $Z$ 并打印出 $F$,$C$,$Z$ 的值。再读第二个数据 135. 5⇒ $F$，再计算 $C$ 和 $Z$ 并行打印相应的 F, $C$,$Z$ 值，直到读最后一个数据 99 赋给 $F$，并计算和打印。打印的结果如下：

```
RUN
F              C                  Z
32             0                  273.16
135.5          57.50001           330.66
68.3           20.16667           293.3267
12.7           -10.72222          262.4378
87.4           30.77778           303.9378
99             37.22223           310.3822
OUT OF DATA IN 20
```

在输出出最后一组数据后，60 语句又使流程转回到 20 语句。但此时 DATA 语句中的数据已全部读完，因此出现“数据区已无数据可读”的错误，打印出“Out of Data”，表示出现了“要读的数据不够”的错误，程序中断运行。但上面输出的所有计算结果仍然是正确的。只是在输出完最后一行后再执行 READ 语句时才出错。因此不影响出错前的计算结果。

上面两个例子是用 GOTO 语句构成无终止循环。这种循环虽能得出结果，但不会自动终止，或者输出出错误信息。它不符合结构化原则，不宜提倡使用。我们所说的循环结构是指能终止的。可以用不同的方法形成循环：

（1） 用 IF 语句和 GOTO 语句形成循环

（2） 用 WHILE 语句形成循环

（3） 用 FOR－NEXT 语句形成循环

## §5.2　用 IF 语句和 GOTO 语句实现循环

**【例 5.3】**　求 $\sum_{n=1}^{10} n$，流程图见图 5.1。

程序为：

```
10    T=0: N=1
```

```
20   T=T+N
30   N=N+1
40   IF N<=10 GOTO 20
50   PRINT " TOTAL="; T
99   END
RUN
TOTAL=55
```

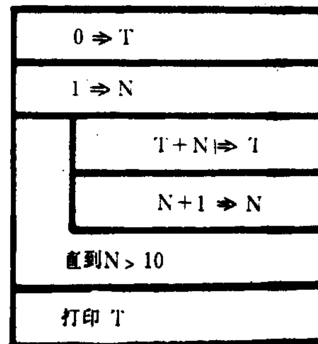



图 5.1

20～40 行构成一个循环。第一次执行完 20 语句后，T 的值为 1，然后 N+1⇒N，N 变成 2。用 40 语句判断是否应继续执行循环。只要 N≤10，就返回 20 语句，继续执行循环。直到第 10 次循环将 N 的值（10）加到 T 中，N+1⇒N，N 的值变成 11，此时 IF 语句中的关系表达式“N<=10”的值为假，不再继续循环。执行 50 语句，输出结果。

注意：(1) N－S 图中，直到型循环的终止条件是“N>10”，而程序中 IF 语句中关系表达式为“N>=10”这两者恰互为“反条件”。这是由于直到型循环结构表示的是“到 N>10 就终止循环”，而 IF 语句表示的是“若 N≤10 时，返回 20 行。继续执行循环”。二者目的一致，仅是表示方法不同。

(2) 程序中 IF 语句中的继续执行循环的条件是“N≤10”，而不是“N<10”，许多初学者往往错写为“N<10”，这样就少进行一次累加操作，未将 N=10 的值累加到 T 中。因为此时 N 虽已等于 10，但并未执行“T+N⇒T”的操作。只有再执行一次循环，将 N 的值加到 T 中，N+1⇒N，此时 N=11，才不应再继续执行循环，这个问题务希弄清楚。

(3) 被反复执行的部分称为“循环体”，本例中 20～30 语句就是循环体，40 语句是循环的控制部分，不属于循环体。

本例也可以改用“当型循环”实现，见图 5.2。

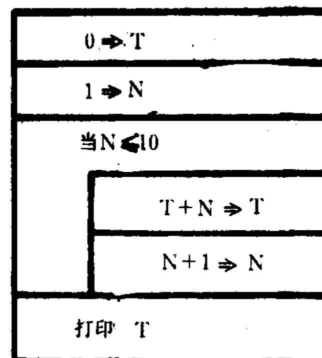



图 5.2

程序可以改为：

```
10 T=0: N=1
20 IF N<=10 THEN T=T+N: N=N+1: GOTO 20
30 PRINT" TOTAL="; T
40 END
```

若循环体内容很多，一个 IF 语句写不下，可以将 20 语句改写为：

```
20    IF N>10 GOTO 50
30    T=T+N
40    N=N+1
45    GOTO 20
```

**【例 5.4】** 对例 5.1 程序作改进，使之能自动终止循环。我们可以人为地设置一个“使循环终止的数据”，当输入此数据时，循环就终止。假设我们以 X=－9999 为控制循环终止的数据，则程序修改为：

```
10   T=0
20   INPUT "X="; X
```

```
30    IF X=-9999 GOTO 99
40    T=T+X
50    PRINT "T="; T
60    INPUT "X="; X
70    GOTO 20
99    END
```

程序运行结果：

```
RUN
X=? 8↙
T= 8
X=? 13↙
T= 21
X=? -12↙
T= 9
X=? 11.52↙
T= 20.25
X=? 9999↙      (不想再累加了，输入 9999，循环结束)
```

这个"使循环终止的数据"称为"终止标志"，它的值可以由程序设计者任选，例如 999，9999，100，10000 都可以，但应注意，它不要与"有用数据"相混淆。若以 99 作为"终止标志"，如果需要将 99 累加到 T 中怎么办？一输入 99，循环就终止了，99 无法累加到 T 中。因此，注意："终止标志"应"远离"有用数据。

**【例 5.5】** 选票的统计。

选票的格式如图 5.3 示，李（Li）同志的代号是 1，张（Zhang）同志的代号是 2，王（Wang）同志的代号是 3。以-1 作终止标志。程序如下：

| 选　　票 | |
|---|---|
| 1 | Comrade Li |
| 2 | Comrade Zhang |
| 3 | Comrade Wang |

图 5.3

```
10 READ A
20 IF A=-1 GOTO 80
30 IF A=1 THEN L=L+1: GOTO 70
40 IF A=2 THEN C=C+1: GOTO 70
50 IF A=3 THEN W=W+1: GOTO 70
60 D=D+1
70 GOTO 10
80 PRINT "COMRADE LI:"; L
90 PRINT "COMRADE ZHANG:"; C
100 PRINT "COMRADE WANG:"; W
110 PRINT "INVALID:"; D
120 DATA 2, 1, 3, 2, 3, 3, 1, 3, 3, 2, 1, 2, 1, 2, 1, 1, 3, 3, -1
999 END
RUN
COMRADE LI: 6
COMRADE ZHANG: 5
COMRADE WANG: 7
INVALID: 0
```

在本例中，由于侯选人没有以“－1”作编号的，因此用“－1”作“终止标志”是可以的。在例 5.3 中，终止标志是从键盘输入的。在本例中，“－1”是放在 DATA 语句中，即写完所应输入的数据后，最后写上标志“－1”。这种用法更为方便。

在本程序中，每次读出 A 值后，首先判断是否“终止标志”，如不是，则判断是否为 1，如是 1，则给李同志加一票（L＋1⇒L）。如不是 1，则再判是否 2，如果是 2，则给张同志加一票（Z＋1⇒C）。如 A＝3，则给王同志加一票（W＋1⇒W）。若读入的不是 1，2，3 之一，则作废票处理（用 D 统计废票），使 D＋1⇒D。30～50 行中，用 GOTO 70 转到 70 语句，然后再返回 10 语句，70 语句作为“汇合点”。当然也可以直接将 30～50 行中的“GOTO 70”改为“GOTO 10”。但程序的清晰度降低了。本程序中各路径全部汇合到 70 语句，再统一进行循环处理。本程序的 N－S 图可画成图 5.4 形式。

通过以上几个例子，可以看到，控制循环的 IF 语句可以出现在循环体的前面（例 5.5），也可以出现在循环体的后面（例 5.4）。希望读者能灵活应用 IF 语句和 GOTO 语句构成循环。应该说明，这二种形式的循环是可以相互转换的。读者可自己完成它。

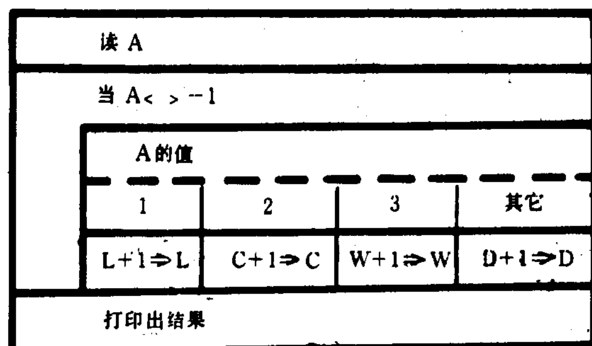



图 5.4

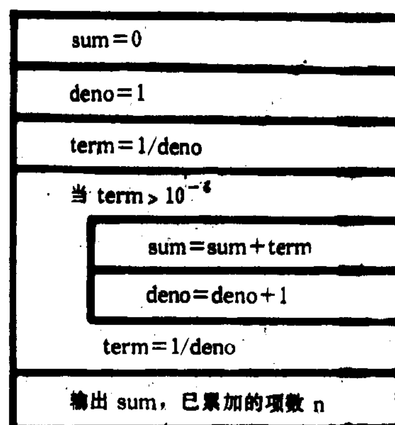



图 5.5

## §5.3 用 WHILE 语句实现循环

MS－BASIC 提供了 WHILE 语句直接实现当型循环，比用 IF－GOTO 实现循环更为方便。它的一般形式如下：

**WHILE〈关系表达式〉**

**<循环体>**

**WEND**

当关系表达式为“真”（即当给定条件满足）时，不断执行循环，直到关系表达式的值为“假”为止。

**【例 5.6】** 求 $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots\cdots+\frac{1}{n}$ 当第 n 项（$\frac{1}{n}$）的值大于 $10^{-4}$ 时，累加一直进行下去，直到 $\frac{1}{n}\leqslant 10^{-4}$ 为止。

算法是比较简单的，见图 5.5

程序如下：

```
10  SUM=0; DENO=1; TERM=1/DENO
20    WHILE TERM >. 0001
30      SUM=SUM+TERM
40      DENO=DENO+1
50      TERM=1/DENO
60    WEND
70  PRINT " SUM="; SUM," N="; DENO-1
99  END
RUN
SUM= 9.787512    N= 9999
```

程序中以 SUM 代表累加和，DENO（denominator 的编写）代表“分母”，以 TERM 代表某一项（$\frac{1}{n}$）的值。先求出第一项（$\frac{1}{1}$）的值。WHILE 语句判断 TERM 的值是否大于 $10^{-4}$，如是，则将 TERM 加到 SUM 中，然后使分母加 1，再求出多项式中下一项的值 TERM. 如此循环下去直到某一次 TERM 的值$\leqslant 10^{-4}$为止（注意该项不加到 SUM 中）。

WEND 是“WHILE END”的缩写，用它标志循环体的范围（WHILE 语句和 WEND 语句之间的语句为循环体。WEND 语句使流程返回 WHILE 语句，判断是否应继续执行循环）。

**【例 5.7】** 若我国人口每年以 1.5%增长率增长，问多少年后我国人口达到或超过 12 亿。1982 年人口数为 10.3 亿。从 1982 年算起。

解此问题的思路是这样的：从 1982 年起，每过一年算一次人口总数：计算公式是：

$$P_1 = P0 \times (1 + R)$$

$P_0$ 为原人口数，$P1$ 为一年后人口数，$R$ 为年增长率。每算完一次都和 12 亿作比较，看是否已达到或超过 12 亿 。画出 N－S 图（图 5.6）。

N 代表从 1982 年起算的年数。据此编出程序：

```
10  R=.015
20  N=0
30  P=1.03E+09
40  WHILE P<1.2E+09
50    P=P*(1+R)
60    N=N+1
70  WEND
80  Y=1982+N
90  PRINT "R=";R,"N=";N,"Y=";Y,"P=";P
99  END
```

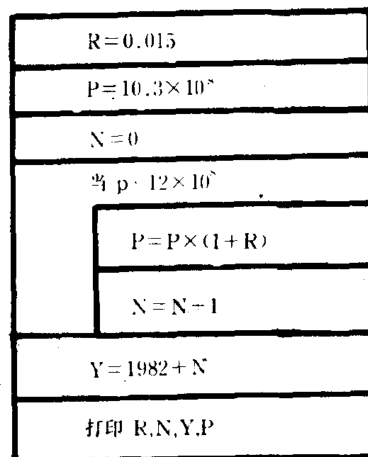



图 5.6

运行结果为：

```
R=.015   N=11   Y=1993   P=1.213287E+09
```

在程序中只设了一个 $P$ 来代表“当年人口数”，50 语句中赋值号右边的 $P$ 是原来的人口数，赋值号左边的 $P$ 是一年后的人口数。不断累增。这样做程序简短些。也可以设两个变量 $P0$,$P1$ 分别代表增长前和一年后的人口数，这时 30～70 语句应改为：

```
30   P0=10.3E8
40   WHILE  P0<=12E8
50   P1=P0*  (1+R)
60   N=N+1
65   P0=P1
70   WEND
```

在求出 P1 之后，还要将 P1 赋给 P0 作为下一年度的“原人口数”。90 语句中的 P 也应改为 P0（用 P1 也可）。

也可以将 10 语句改为：

10 INPUT “R=”； R

由键盘输入年增长率。这样可以计算不同增长率的情况。

**【例 5.8】** 给一个数 *M*，判断它是否素数（即质数）。

所谓素数，指除了 1 和它本身之外，不能被任何整数整除的数，例如 17 是素数，因为它不能被 2～16 间任何整数整除。判断 M 是否素数最原始的方法就是先用 2 除 M，再以 3 除 M，……，最后以 M－1 除 M，如果都除不尽，则 M 必然是素数。只要其中有一次被除尽，则 M 不是素数。其实 M 不必被 2～（M－1）间全部整数除，只要被 2～$\sqrt{M}$（如$\sqrt{M}$不是整数则取其整数部分）除即可。例如判断 17 是否素数，只需将 17 被 2，3，4 除即可（$\sqrt{17}$取整得 4）。为什么？请读者分析。

在考虑算法时，要考虑：循环的条件是什么？①除数 I 由 2 开始，当除数≤$\sqrt{M}$时，循环就要继续下去；②当某一次 M 被 I 整除，则不应再循环下去。怎样表示“M 能被 I 整除”呢？可以设一个“开关”Switch，当它的值等于 0 时代表“尚未被 I 整除过”，一旦被某一个 I 整除，就使它的值为 1。因此，循环条件应为“I≤$\sqrt{M}$和 Switch=0”。算法见图 5.7。

当在某一次循环中，M 被 I 整除，switch 的值变成 1，根据循环条件，不再执行循环体。注意图 5.7 中 $S_5$ 这一步。若 M 曾被任一个 I 整除，则 *Switch* 的值必为 1，因此打印 *m*“不是素数”。若 M 未曾被任一 I 整除，则 *Switch* 保持为 0。在 $S_5$ 步骤中打印 M“是素数”。

程序：

```
10   INPUT" M=", M
15   IF M=2 THEN PRINT " 2 is a prime number": END
20   N=INT (SQR (M))
30   I=2: SWITCH=0
40   WHILE I<=N AND SWITCH=0
50     IF M/I=INT  (M/I) THEN SWITCH=1 ELSE I=I+1
60   WEND
70   IF SWITCH=0 THEN PRINT M;" is a prime number"
     ELSE PRINT M;" is not prine number"
90   END
```

运行结果如下：

①M=17↙

17 is a prime number

②M=16↙

18 is not prine number

③M=2↙

2 is a prime number

请注意 50 语句中判定 M 是否能被 I 整除的方法。如果下式成立，则 M 能被 I 整除。

M/*I* =INT（*M*/*I*）

例如：*M* =16，*I* =4，16/4=INT (16/4) 成立，等号两侧都等于 4，INT *M* 能被 *I* 整除。如果 *M* =17，*I* =3，17/3≠(17/3)=5，17 不能被 3 整除。这是因为，若一个数 X 等于它的整数部分（INT (X)），则 X 必为整数，例如 6=INT（6），而 6.5≠INT（6.5）。

求两个数相除的余数的方法是：

R=M−INT（M/I）＊I

M 和 I 都是整数。例如求 17 被 3 除的余数 R。R=17−INT（17/3）＊3=17−INT（5.666666）＊3=17−5＊3=2。若 R=0，表示 M 能 I 整除，因此，50 语句也可改写为：

50 IF M−INT（M/I）＊I=0 THEN SWITCH=1 ELSE I=I+1

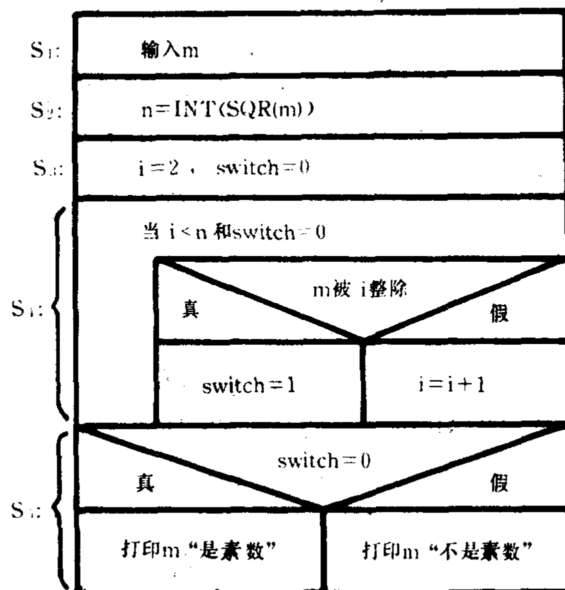



图 5.7

有的计算机系统所用的 BASIC（如 Applesoft BASIC）不提供 WHILE-WEND 语句，可以改用 IF-GOTO 语句来实现当型循环。本节中几个例子都可以改用 IF-GOTO 来实现，请读者自己完成它，并不难。请读者仔细消化这几个例子中的算法，它们都是很基本的。

## §5.4 用 FOR-NEXT 语句实现循环

如果已知循环次数的话，使用 FOR-NEXT 语句实现循环更为方便。先看下面两个程序。

**【例 5.9】**

程序（a）

```
10    X=1
20    IF X>5 GOTO 60
30    PRINT "X="; X
40    X=X+1
50    GOTO 20
60    END
RUN
X=1
X=2
X=3
```

程序（b）

```
10    FOR X=1 TO 5 STEP 1
30    PRINT "X="; X
40    NEXT X
60    END
RUN
X=1
X=2
X=3
```

X＝4　　　　　　　　　　X＝4

X＝5　　　　　　　　　　X＝5

程序（a）是利用 IF－GOTO 语句编程序，显示 X 的值，每次循环后 X 的值加 1，共循环 5 次。程序（b）是用 FOR－NEXT 语句实现上述要求的，也是循环 5 次。它们的作用是相同的，都可以用图 5.8 表示其流程。可以看出，用 FOR－NEXT 语句实现循环，能使 X 自动得到一个初值，在每次循环后 X 自动增值，在每次循环开始时将 X 与 5 进行比较，如果 X＞5 就终止循环。显然，程序（b）比程序（a）使用更加方便。

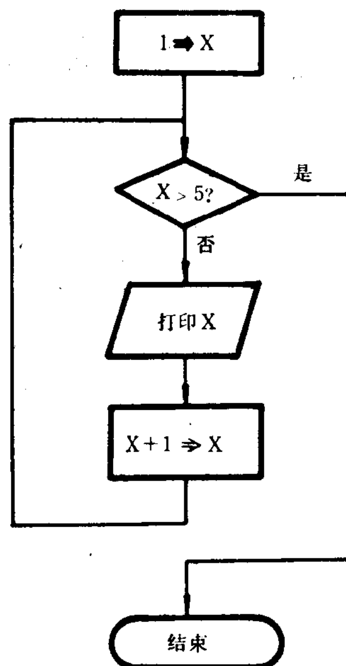



图 5.8

## 5.4.1　循环语句的结构

FOR 语句是循环的起点，称“循环说明语句”或“循环起始语句”。由它决定应执行几次循环。NEXT 是“循环终端语句”。FOR 和 NEXT 语句两者必须成对出现，缺一不可。FOR 语句中的变量（称循环控制变量）必须与 NEXT 语句中的变量相同（如上例中的 X）。

FOR 语句的一般形式为：

**FOR X＝A TO B STEP C**

X 为循环控制变量（或称循环变量），可以用任一简单变量来表示；

A 为循环变量初值；

B 为循环变量终值；

C 为循环变量的增量（或步长）。

这个语句的意思是：循环变量 X 取值从 A 开始，到 X 超过 B 时结束。每循环一次 X 的值增加 C。

A、B、C 可以是常数，或已被赋值的变量，也可以是算术表达式（表达式内的变量必须已被赋值）。如：

10 FOR X＝3＊5 TO LOG（20）STEP－5/2

NEXT 语句的一般形式是：

**NEXT X**　　　　（X 为循环变量）

NEXT X 的意思是：执行到此语句时，循环变量就取下一个 X 值，即加一个步长值。然后判断和决定应否进行下一次循环。

### 5.4.2 循环语句的执行过程

(1) 在执行 FOR 语句时，把初值 A 赋给循环变量 X，并将终值与步长记下来。将 X 值与终值 B 比较。如果 X 超过 B 则不执行 FOR 与 NEXT 两语句之间的所有语句(这些语句称循环体)，而跳到 NEXT 语句的下面一句。即“跳过”循环体。

(2) 如果循环变量的值 X 未超过终值 B，则顺序执行 FOR 与 NEXT 两语句之间的语句。循环变量一直保持原来的值直到 NEXT 语句（除非重新对它赋值)。

(3) 执行到 NEXT 语句时，循环变量自动按“步长”增值，“步长”由 STEP 后面的数值给定。也就是 X+C⇒X（在上面的例子中第一次执行 30 语句时，X+1⇒X，即将 1+1⇒X，X 值变成 2)。

(4) 将已增值的循环变量 X 再与终值 B 比较，如 X 仍未超过终值，则重复执行 (2)、(3) 和 (4)，每循环一次，循环变量 X 的值增加一个步长值 C。如此反复循环，直到循环变量的值超过终值为止。此时转而执行 NEXT 后面的语句。

例 5.9 中程序 (b) 在 IBM－PC、长城 0520 和 APPLE－Ⅱ 计算机上运行情况如下：

| 第几次循环 | X | 与终值相比 | 执行循环体否 |
|---|---|---|---|
| 1 | 1 | <5 | 执行 |
| 2 | 2 | <5 | 执行 |
| 3 | 3 | <5 | 执行 |
| 4 | 4 | <5 | 执行 |
| 5 | 4 | =5 | 执行 |
| 6 | 5 | >5 | 停止执行 |

当 $X=5$ 时，还要执行循环体（打印出“5”）一次，当再执行到 NEXT 语句使 $X=6$ 时，就停止执行循环而执行 60 语句（END 语句)。共循环了五次。

循环的流程图见图 5.9。

### 5.4.3 执行 FOR 循环中的一些问题

(一) **步长值可以为正值**（循环变量递增)，**也可以是负值**（递减)，即 STEP 之后的 C 可以是正数或负数。当步长值为 1 时，“STEP”和它后面的步长值可以省略不写。

**【例 5.10】** 步长为正值。

```
10   FOR X=1 TO 10 STEP 2
20   PRINT X;
30   NEXT X
```

```
40    END
RUN
1  3  5  7  9
```

循环共 5 次，当执行完 5 次后，$X$ 又加 2，$X$ 的值等于 11，而 11>10（终值），故脱离循环而执行 40 语句。当步长为正值时，循环变量的值在变到大于终值时才终止循环。

图 5.9

【例 5.11】 步长为零。

```
10    FOR X=1 TO 10 STEP 0
20    PRINT X;
30    NEXT X
40    END
RUN
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1……
```

一直循环下去，因为 $X$ 值始终等于初值 1 而小于终值 10。

【例 5.12】 步长为负值。

```
10    FOR X=10 TO 1 STEP -1
20    PRINT X;
30    NEXT X
40    END
RUN
10  9  8  7  6  5  4  3  2  1
```

每经过一次循环，X 值就减少 1，循环 10 次后 $X=1$，由于步长为负值，循环变量是从一个大的数值“走向”一个小的数值，$X=1$ 时，它还没“走过”终值 1，因此，还要再执行一次循环体，打印 $X$ 值。当 X 再加一次 −1，$X=0$ 了，此时它已“超过”终值 1 了，脱离循环。

可以用图 5.10 示意。

(a)

(b)

图 5.10

在图 5.10（a）中初值（−10）小于终值（0），步长为正。X 由小到大变化，当 X>终值 0 时，循环停止。

在图 5.10 (b) 中初值 (10) 大于终值 (0) 步长为负。X 由大到小变化，当 X <终值 0 时，循环停止。

因此可以说：当循环变量沿变化的方向“超过”终值时，就停止循环。

当步长为负时，循环变量必须小于终值才停止循环。

**【例 5.13】** 初值、终值、步长均为负值。

```
10   FOR X=-100 TO -200 STEP -20
20   PRINT X;
30   NEXT X
40   END
RUN
-100  -120  -140  -160  -180  -200
```

在循环六次后，X 变成－220，小于终值－200，停止循环。

当步长为负时，图 5.9 的框图中的判断框（菱形框）内的“X 超过终值?”可以理解为“X<终值?”。

步长值不一定是整数，可以是小数。如：

```
10   FOR X=1 TO 4.5 STEP 0.5
20   PRINT X;
30   NEXT X
40   END
```

请读者自己写出运行结果。

（二）**在循环体内可以出现循环变量**（如上例中的 X），此时循环变量的值在每次循环中是不同的。循环变量也可以不在循环体内出现，此时，循环变量的作用仅在控制循环的次数。

**【例 5.14】**

```
10   LET X=1
20   FOR I=1 TO 5
30     PRINT X;
40   NEXT I
50   END
RUN
 1  1  1  1  1
```

20 语句的作用就是把每次的 I 值与终值 5 相比，当循环了五次后，I=6 超过 5，就不再执行循环了。

下面两个程序作用完全相同：

**【例 5.15】**

程序 (1)

```
10   FOR I=1 TO 5
20     READ X
30     PRINT X;
40   NEXT I
50   DATA 1, 2, 3, 4, 5
```

程序 (2)

```
10   FOR I=6 TO 10
20     READ X
30     PRINT X;
40   NEXT
50   DATA 1, 2, 3, 4, 5
```

```
60  END
RUN
 1  2  3  4  5
```

在程序（2）中并不因 I 由 6 变到 10 而在运行时打印出 6，7，8，9，10 来。它的作用仅仅在控制循环次数（当步长为 1 时，从 6 到 10 应循环五次）。

如果把上述程序中的 10 语句改为：

```
10 FOR I=0.1 TO 0.5 STEP 0.1
```

请读者自己分析一下它的运行结果是什么？

（三）**在循环中一般不应再对循环变量赋值**。如果有以下程序：

**【例 5.16】**

```
10  FOR X=1 TO 5
20    X=2*X
30    PRINT X,
40  NEXT X
50  END
```

请问，执行几次循环体呢？如果没有 20 语句显然应执行 5 次循环体。现在由于 20 语句使循环变量的值改变了。在执行第一次循环时 X 初值为 1，执行 20 语句后 X 的值变为 2，执行 40 语句后 X 再增值变为 3。在执行第二次循环时，X 的值开始时为 3，执行 20 语句后 X 的值变为 6，在打印 X 的值 6 之后，循环终止，共执行了两次（而不是 3 次）循环。见表 5.1。

5.1

| 第几次循环 | 执行本次循环开始时 X 的值 | 执行 20 语句后 X 的值 | 执行 40 语句后 X 的值 | 与终值相比较 | 循环终止否? |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 3<5 | 未 |
| 2 | 3 | 6 | 7 | 7>5 | 终止 |

程序编写者的原意是想输出出 2，4，6，8，10 这样五个数，结果只能输出两个数。

```
RUN
 2        6
```

因此，在一般情况下不提倡对循环变量再赋值。但在某些时候，也可以利用以上特性在未达到应循环次数时提前终止循环。

**【例 5.17】** 本章例 5.9 求素数也可以用下面的程序实现：

```
10 INPUT"M=", M
15 IF M=2 THEN PRINT"2 is a prime number": END
20 N=INT (SQR (M))
30 FOR I=2 TO N
40 IF M/I=INT (M/I) THEN I=2*N
60 NEXT I
70 IF I<=2*N THEN PRINT M;" is a prime number" ELSE PRINT M;" is not a prine number"
90 END
```

如果在某次循环中 M 被 I 整除，使 I=2N，在执行完 60 语句后，I 的值为 2N+1。70 语句判别 I 是否>2N，如果 I>2N 表示“M 曾被 I 整除”。如果 M 都不曾被任何一个 I 整除，在脱离循环时 I 的值为 N+1，它必然小于 2N。据此来判断 M 是否素数。请读者仔细消化此程序。

## §5.5 循环语句应用举例

**【例 5.18】** 求 5!，即 1×2×3…×5

```
10   LET T=1
20   FOR I=2 TO 5
30     LET T=T*I
40   NEXT I
50   PRINT T
60   END
RUN
 3.6288E+06
```

用变量 $T$ 存放每次累乘后的结果，乘数 $I$ 由 2 变化到 5，即将 1×2，再乘 3，乘 4，…，乘到 5 为止。

**【例 5.19】** 求 $\sum_{n=1}^{20} n!$（即求 1+2!+3!+4!+…20!）

```
10   LET S=0
20   LET T=1
30   FOR N=1 TO 20
40     LET T=T*N
50     LET S=S+T
60   NEXT N
70   PRINT S
80   END
```

请读者自己写出每次循环时的 N、T、S 值。

**【例 5.20】** 相传古代印度国王舍罕要褒赏他的聪明能干的宰相达依尔（国际象棋的发明者），问他需要什么，达依尔回答说：“国王只要在国际象棋的棋盘第一个格子上放 1 粒麦子，在第二个格子上放 2 粒，第三个格子上放 4 粒，以后按此比例在下一格中加一倍的麦子，一直放到第 64 格（国际象棋是 8×8=64 格，见图 5.11），我就感恩不尽了，其它什么我都不要了”。国王想：“这有多少！还不容易！”，让人扛来一袋小麦，但不到一会全用没了，再来一袋又很快用没了，结果全印度的粮食全部用完还不够。国王纳闷，怎样也算不清这笔账。现在我们用计算机来算一下。

我们需要计算的是：$T=1+2+2^2+2^3+\cdots 2^{63}$ $T$ 是小麦的颗粒。1 立方米约有 $1.42\times10^8$ 颗。设 $S$ 为体积。

程序为：

```
10   FOR N=0 TO 63
20   P=2∧N
```

```
30    T=T+P
40 NEXT N
50 PRINT " total ="; T
60 S=T/1.42E+08
70 PRINT " Volume ="; S
99 END
RND
total=1.844674E+19
Volume= 1.299067E+11
```

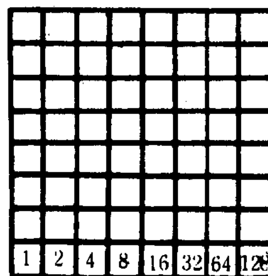


图 5.11

计算结果为：共有小麦 1.84467×$10^{19}$ 颗，体积约 1.3×$10^{11}$$m^3$。它相当于全中国 960 万平方公里的土地上，全铺满约 1.3cm 厚的小麦粒。它相当于我国几百年的产量。

**【例 5.21】** 假设有一对兔子，到第三个月又生下一对兔子。设以后每对兔子到第三个月都又生一对兔子，问到第 50 个月共有多少对兔子。这是一个古典的教学问题。可以推算出，各月的兔子对数依次如下，1，1，2，3，5，8，13，21，……。即第一二个月只有一对，第三个月二对，第四月 3 对，第五个月 5 对……。可以发现这样一个规律：从第三个月起，兔子对数为其前二个月兔子对数之和，可以用数学方法表述如下：

一个数列，其头二个数为 1，1，从第三个数起每个数都为其前二个数之和。这个数列称为“斐波那契（Fibonacci）数列”。今求该数列前 50 个数。

```
10 A=1; B=1
20 FOR I=1 TO 25
30      PRINT A, B,
40      A=A+B
50      B=B+A
60 NEXT 1
70 END
RUN
```

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 5 |
| 8 | 13 | 21 | 34 | 55 |
| 89 | 144 | 233 | 377 | 610 |
| 987 | 1597 | 2584 | 4181 | 6765 |
| 10946 | 17711 | 28657 | 46368 | 75025 |
| 121393 | 196418 | 317811 | 514229 | 832040 |
| 1346269 | 2178309 | 3524578 | 5702887 | 9227465 |
| 1.493035E+07 | 2.415782E+07 | 3.908817E+07 | 6.324599E+07 | 1.023342E+08 |
| 1.655801E+08 | 2.679143E+08 | 4.334944E+08 | 7.014086E+08 | 1.134903E+09 |
| 1.836312E+09 | 2.971215E+09 | 4.807527E+09 | 7.778741E+09 | 1.258627E+10 |

从图 5.12 中可以看到：A 和 B 初始值都为 1，它们代表第一、二个数。在执行第一次循环体时，40 语句使 A+B⇒A，A 的值变为 2，50 语句使 B+A⇒B，注意此时 A 的值已变为 2 了，因此 B 的新值为 1+2=3。以后每次执行循环体时，A 和 B 各得到一次新值。每一次打印出两个数，故应循环 25 次。

这类问题属于“递推”(recurence)问题，即后面的值要从前面的值才能推算出来。譬如问你数列中第13个数是多少？你无法用一个公式直接算出来，必须先算出11和12个数才能知道第13个数。解决递推问题必须先确定“递推关系”(或称“递推公式”)。对Fibonacci数列来说，其递推关系如下：

$F_n=F_{n-1}+F_{n-2}\qquad(n>2)$

其初始条件为：

$F_1=1\qquad F_2=1$

以$F_n$代表数列中第几个数，数列中各数的值可以表示如下：

$$\begin{cases}F_n=1 & (n\leqslant 2)\\ F_n=F_{n-1}+F_{n-2} & (n>2)\end{cases}$$

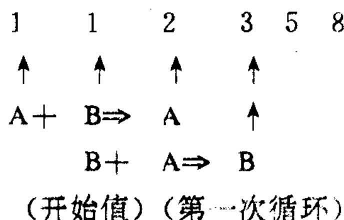


图 5.12

在用计算机解递推问题时，常用循环来处理，计算机运算速度快，处理这类问题得心应手。关键是找出递推关系。在用循环处理时，不可能分别用许多变量名代表$F_1$，$F_2$，$F_3$……$F_n$，我们在程序中只用了两个变量A和B，它们的值不断变化，在第一次循环中它们代表$F_1$，$F_2$，在第二次循环中代表$F_3$，$F_4$，在第三次循环中代表$F_5$，$F_6$，……。可知，A和B不断地以新值代替其原值。这种不断地以新值代替原值的操作称为迭代(iterate)。程序中的A，B称为“迭代变量”。它们的值是不断更迭的。

递推的问题一般可以用迭代方法来处理。如本例。也可以不用迭代方法处理，例如可以设许多变量F1，F2，F3，…，采取F1+F2⇒F3，F2+F3⇒F4，…这是递推，但不是迭代，因为没有一个变量是存在以新值代替该变量的原值的。请区分“递推”与“迭代”的概念。用循环实现迭代算法是很方便的。

**【例 5.22】** 求图5.13所示AB两端的等效电阻R。

可以采取由右向左逐级递推的方法，把整个电路用虚线分为若干级。在第一级中，串联电阻值为：$R_1+R_0$，把这个值送到变量R中(即使$R=R_1+R_0$)，再以R与$R_2$并联，并联值为：$\frac{R\times R_2}{R+R_2}$，再把它送到R，即$\frac{R\times R_2}{R+R_2}\Rightarrow R$，注意双箭头左边的R是R的原值，它是$R_1+R_0$。箭头右边的R是R的新值。这时求出的R是第一级的等效电阻。再以这个R与第二级中的串联电阻$R_1$相加，然后再与电阻$R_2$并联，求出第二级等效电阻R值，如此一直求出总的等效电阻R值。

用这种办法解，程序是很简单的：

```
10 INPUT "R0,R1,R2=",R0,R1,R2
20 PRINT " I"," R"
30 R=R0
40 FOR I=1 TO 5
50     R=R+R1
60     R=R*R2/(R+R2)
70     PRINT I, R
80 NEXT I
90 END
```

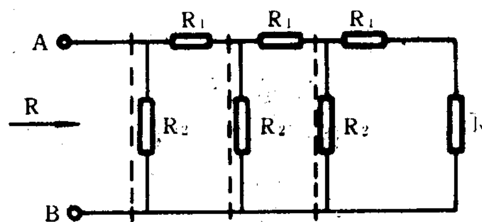


图 5.13

程序中用了迭代方法，R是迭代变量，运行后，它把每一级的等效电阻都打印出来：

```
RUN
R0, R1, R2=100, 200, 300↙
 I          R
 1          150
 2          161.5385
 3          163.9535
 4          164.4483
 5          164.5493
```

# §5.6 循环的嵌套

单层循环可以解决一些简单的问题，但实际上有许多问题需要用两层甚至多层循环才能解决。在一个循环中又包含一个循环称为循环的嵌套。

**【例 5.23】** 打印乘法九九表。即 1×1=1，1×2=2，……，9×9=81。

我们先考虑如何打印 1×1=1，1×2=2，…… 1×9=9。这用一个单层循环就可以解决。

```
10 A=1
20 FOR B=1 TO 9
30     LET P=A*B
40     PRINT A; "*"; B; "="; P
50 NEXT B
60 END
```

如果要打印 2×1=2 到 2×9=18，则要改 10 语句，使 A=2。

要打出整个“九九表”，则要使 A 先后等于 1，2，3，……9。每个 A 值都要执行一次 20 语句到 50 语句的 B 循环（即 A 每次固定为一个值，B 从 1 变化到 9，打印出相应的乘积）。

用双重循环就可使问题简单多了：

```
10 FOR A=1 TO 9
20     FOR B=1 TO 9
30     LETP=A*B
40     PRINT A; "*"; B; "="; P
50     NEXT B
60 NEXT A
70 END
```

把它和前面那个程序比较一下；只修改了 10 语句，增加一个 60 语句，形成一个双重循环。它的运行结果是打印出从 1×1=1 到 9×9=81 的整个“九九表”。

程序的流程图见图 5.14。N－S 图见图 5.15。

这个双重循环的执行过程是这样的：

(1) 把初值赋予循环变量 A，即 1⇒A（这个循环变量 A 的值一直保持到遇到外循环的 NEXT A 语句才变化）。然后开始执行外循环的循环体，即外循环中 FOR 语句和 NEXT 语句之间的各语句（20 语句到 50 语句），而 20 语句到 50 语句又是一个内循环，在 A=1 时，B 从 1 变化到 9，30 和 40 语句被执行 9 次，打印出 1×1=1 到 1×9=9。

(2) 当内循环执行完后 (B>9), 程序执行 60 语句, A 增值, 由 1 变成 2, 由于 A<9, 程序又重新执行 20 语句到 50 语句(但此时循环变量 A 的值已改变为 2 了)。计算机打印出 2×1=2 到 2×9=18。

(3) 60 语句再使 A 增值, A=3。再执行 20 语句到 50 语句。如此反复, A 值由 1 依次变到 9, 最后一次打印出 9×1=9 到 9×9=81。外循环完, 结束。

应当注意的是: 当每次外循环变量 A 的值改变时, 内循环变量 B 必须重新取初值, 从流程图上可以清楚地看出这一点。

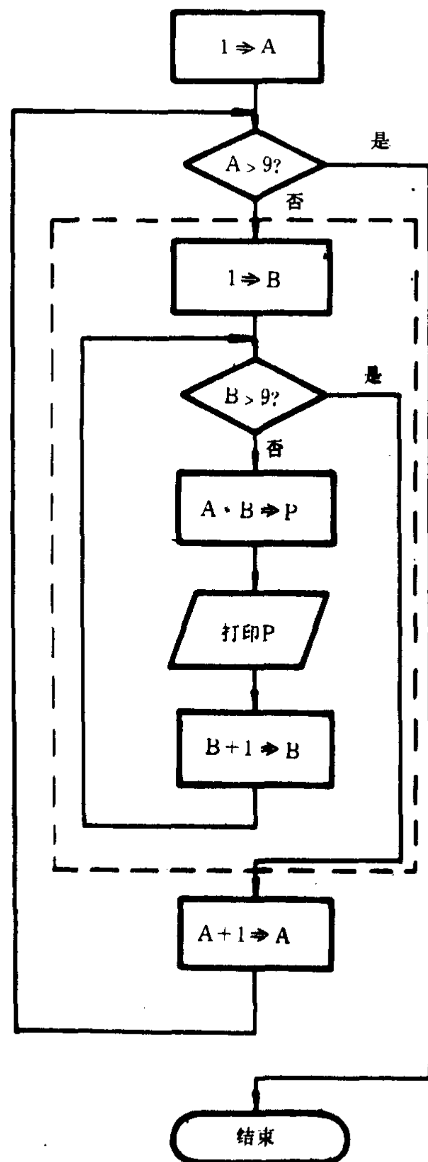


图 5.14

在一个循环中又包含另一个或多个循环, 叫做循环嵌套。上面的例子是两个循环的嵌套。图 5.13 中虚线框是内循环, 它又是外循环的循环体。

**【例 5.24】** 统计五个班的每班平均成绩, 并分别打印出来, 每个班的学生为 10 人, 显然这是一个只能用双层循环才能处理的问题。程序是很简单的。

```
10   FOR I=1 TO 5
20   S=0
30   FOR N=1 TO 10
40   READ G
50   S=S+G
60   NEXT N
70   A=S/10
80   PRINT" class"; I," average="; A
90   NEXT 1
100  DATA 65, 78, 96, 100, 57, 85, 77, 89, 98, 60
110  DATA 78, 65, 61, 54, 30, 99, 100, 75, 58, 61
120  DATA 100, 90, 81, 72, 63, 74, 88, 89, 51, 80
130  DATA 92, 91, 80, 71, 88, 75, 59, 61, 85, 73
140  DATA 53, 60, 62, 70, 89, 77, 44, 81, 83, 99
150  END
RUN
class 1      average= 80.5
class 2      average= 68.1
class 3      average= 78.8
class 4      average= 77.5
class 5      average= 71.8
```

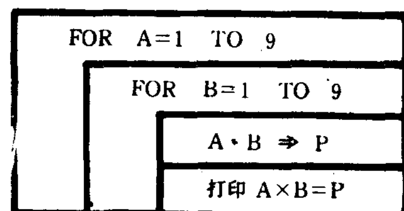


图 5.15

循环的嵌套既可以是 ROR 循环与 FOR 循环的嵌套, 也可以是 FOR 循环与 WHILE 循环(或用 IF-GOTO 语句构成的循环)嵌套。

**【例 5.25】** 求出 3～100 之间的全部素数

在例 5.8 和例 5.17 中已解决了如何判断一个数是否素数的问题。现在只需在原有程序

之外加一层外循环，使 M 由 3 变到 100 即可。

程序如下：

```
10    FOR M=3 TO 100 STEP 2
20    N=INT (SQR(M))
30     FOR I=2 TO N
40     IF M/I=INT (M/I) THEN I=2*N
50     NEXT I
60    IF I<2*N THEN PRINT M;
70    NEXT M
80  END
RUN
5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73  79
83  89  97
```

由于偶数显然不可能是素数，因此 10 语句中用“STEP 2”，使 M 只取奇数。题目只要求打印出素数，对非素数不打印任何信息。

关于多重循环的一些规定：

（1）某些计算机对循环的嵌套层数有所限制，例如最多不得超过八层，超过时按出错处理；而有的并不限制。因此，使用时最好查阅你所用计算机的 BASIC 使用手册。实际上，一般程序中很少超过八层循环嵌套的。因此，可以放心使用嵌套。

（2）在循环嵌套中，内外循环不得交叉。如：

```
FOR
FOR     X          FOR     X
: :     Y          FOR     Y
NEXT               : :
NEXT    Y          NEXT    X
        X          NEXT    Y

正确               错误
```

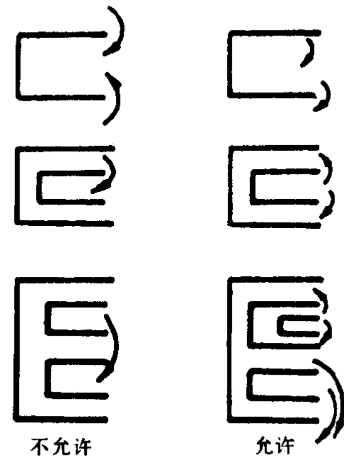



图 5.16

（3）在循环中可以用转向语句使流程从循环体内转到循环体外，但不允许从循环体外转入循环体内。如果是多重循环，则允许从内循环转到外循环体，不允许从外循环转入内循环体。见图 5.16。

下面程序是合法的：

```
10  FOR I=1 TO 5
20  IF I=3 GOTO 50
30  PRINT I
40  NEXT I
50  END
```

但它不符合结构化原则，因为对这个循环来说，有一个入口，两个出口。建议尽可能不要从循环体中转出去。

下面程序是不合法的：

```
10    GOTO 30
20    FOR I=1 TO 5
30    PRINT I
40    NEXT I
50    END
```

循环结构和选择结构是十分有用而重要的，几乎所有的实用程序都包含这两种结构，因此我们在本章中介绍了比较多的例子，希望大家能深入掌握。在本书第十三章“常用算法程序举例”中还介绍了一些较深入的算法，读者在学习本章时可以从中选择一些例题阅读。

## 习　题

用 IF - GOTO 语句和 WHILE 语句编写程序，处理 5.1～5.6 题。

5.1　求 $n!$，$n$ 由键盘输入

5.2　求 $\sum_{n=1}^{10} n!$

5.3　用 $\frac{\pi^2}{6}=\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\cdots\cdots+\frac{1}{n^2}$ 近似公式求 $\pi$ 值，当 $\frac{1}{n^2}<10^{-5}$ 时不再累加。

5.4　编写一个“加法器”的程序。用 INPUT 语句从键盘输入若干个数据（数据个数和数值每次不同）。要求打印出：数据的个数，数的总和。（用“终止标志”）

（1）数据为：8，16.5，－13，71.5，43.7

（2）数据为：－2，42.1，46.2，－13.6，55，72，11

5.5　用近似公式求自然对数的底 $e$ 的值。$e\approx 1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots\cdots+\frac{1}{n!}$ 直到 $\frac{1}{n!}<10^{-5}$ 为止。

5.6　一个工厂的产值以每年 10%增长，问多少年后产值可以翻一番。

5.7　写出下面各程序运行时打印的结果。

（1）
```
10    FOR I=15 TO 25 STEP 3
20    PRINT I,
30    NEXT I
40    END
```

（2）
```
10    FOR X=-10 TO -100 STEP -20
20    PRINT X,
30    NEXT X
40    END
```

（3）
```
10    A=3
20    B=6
30    FOR Y=B-A TO A+B STEP B/A
40    PRINT A, B
50    NEXT Y
60    END
```

（4）
```
10    T=5
20    FOR N=3 TO T STEP 0.5
```

```
30  PRINT T * T
40  NEXT N
50  END
```

5.8 分析下面这个程序，当这个程序运行后会产生下面结果中的哪一个？

```
10  LETN=1
20  FOR K=1 TO N
30  PRINT "*";
40  NEXT K
50  PRINT
60  LETN=N+1
70  IF N<=8 GOTO 20
99  END
```

```
       *
      ***
     *****
    *******
   *********
  ***********
 *************
***************
```

(1)

```
********
 ********
  ********
   ********
    ********
     ********
      ********
       ********
```

(2)

```
*
**
***
****
*****
******
*******
********
```

(3)

5.9 计算 $Y=\frac{x^2}{1!}+\frac{x^3}{2!}+\frac{x^4}{3!}+\frac{x^5}{4!}+\cdots+\frac{x^{N+1}}{N!}$

设 $x=2$. 要求打印出 N 分别等于 1，2，3，4，5，6，7，8，9，10 时 $Y$ 的值。

5.10 有任意十个数，请把它们之中最大的和最小的数打印出来（十个数由自己设定）。

5.11 一个数列，它的头三个数是 0，0，1。第四个数是前三个数之和，以后每个数都分别是其前三个数之和。请编出此程序，打印出此数列，直到第 60 个数或数的值达到（或超过）$10^{15}$ 为止。

5.12 求：当我国人口增长率为 3%，2. 5%，2%，1. 5%，1%，0. 5%时，从 1982 年算起多少年就会达到或超过 12 亿。1982 年人口为 10.3 亿。要求用 DATA 语句放以上的各个增长率。在读完数据后，用“终止标志”处理。(用双层循环)。

5.13 统计五个班的每班平均成绩，并分别打印出来，设每个班学生人数不相等，但都不超 10 人，请编程序。

5.14 求 $\sum_{k=1}^{100}k+\sum_{k=1}^{50}k^2+\sum_{k=1}^{10}\frac{1}{k}$

5.15 有一光滑斜面，与水平桌面成 α 角，设有一质点在 $t=0$ 时从此斜面的顶点 $A$ 开始由静止状态自由释放（见图 5.17）。如果忽略摩擦力，斜面的长度 $s=300$cm，α=65°，求 t=0.1，0.2，0.3，……1.0 秒时质点的速度。

5.16　给 $m,n$ 两个整数，求它们的最大公约数和最小公倍数。

5.17　给一个正整数，求出它的因数，并按下面式样打印出来。例如：15=3*5，20=2*2*5，56=2*2*2*7

5.18　打印出所有的水仙花数，所谓水仙花数是指一个三位数，其各位数字立方和等于该数。例如，153 是一水仙花数，因为 $153=1^3+5^3+3^3$。

5.19　一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如，6 的因子为 1，2，3，6=1+2+3，所以 6 是一个完数，编程序找出 1000 之内的所有完数，并打印出它们的因子，如：

6 ITS FACTORS ARE 1　2　3

5.20　解数学灯迷。有以下算式：

$$\begin{array}{r} A\ B\ C\ D \\ -\quad C\ D\ C \\ \hline A\ B\ C \end{array}$$

$A,B,C,D$ 均为一位非负整数，要求找出 $A,B,C,D$ 各值

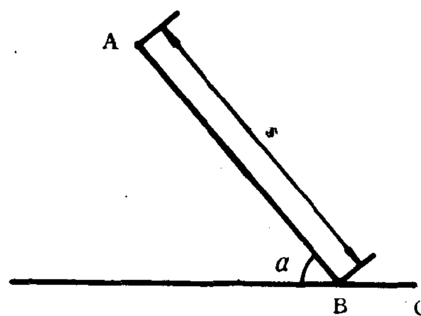


图 5.17

5.21　有一分数序列：

$$\frac{2}{1},\ \frac{3}{2},\ \frac{5}{3},\ \frac{8}{5},\ \frac{13}{8},\ \frac{21}{13},\ \cdots\cdots$$

求出这个数列前 20 项之和。

# 第六章　数　　组

## §6.1　数组和数组元素的概念

至今为止，我们讲过的变量都属于简单的变量，如 $A$、$B$、$C$ 等。可以给一个简单变量赋予一个数值。但是常常遇到这样的情况：对一个物理对象，要进行多次测量。如在物理试验中测电流 $I$，共测 $n$ 次，然后将每次测出来的值相加，再将其和除以 $n$，可得到电流的平均值。如果用 $n$ 个简单变量来代表 $n$ 次测出的值，这样是不太方便的。实际上我们每一次测的都是电流，仅仅是每一次测出的数值不同而已。我们可以把这几个同一类型的变量，用同一个名字来代表，例如用 $I_1$，$I_2$，……$I_n$ 来代表 $n$ 次测出的电流。这个 I 就是代表一组电流值的“数组”，右下角的数字 1，2，……n 称 I 的下标。$I_n$ 表示第 n 次测出的电流值。

用一个统一的名字来代表具有相同属性的一组数，这就是数组名字的来由。数组中各个数称为数组元素。为了确定各数据与数组中每一元素的一一对应关系，必须给数组中这些数编号，即顺序号。用下标来指出顺序号。例如，I 数组中第五个元素就代表第五次测出的电流值。因此，可以说，数组是一批有序数据的集合，或者说，数组是用一个名字代表顺序排列的一组数（简单变量是没有序的，无所谓谁先谁后，而数组中的元素是有排列顺序的）。顺序号就是下标的值。

在 BASIC 中，要引用数组中某一元素，应指出数组名和用括弧括起来的下标，例如：

I（7）　　（相当于数学中的 $I_7$，其中 I 为数组名，7 为下标值）

I（7）代表 I 数组中顺序号为 7 的那个元素。数组元素（例如 I（1），I（5）……）又称“带下标的变量”，简称“下标变量”。

又如，全班有 30 个学生，在 BASIC 中 30 个学生的成绩可以用 S（1），S（2）……S30）来表示。

总结起来，我们已学过两种变量，即简单变量（如 A，X1，X5）和下标变量（如 I（1），S（30））。

在 BASIC 中有关数组和下标，要作以下几点说明：

**（一）数组名的规则。**与简单变量的名字规则相同，即由若干个字母和数字组成，并且第一个为字母，如 ABC，AMOUNT，AGE，COURSE 都可以作为合法的数组名。

**（二）下标必须用括弧括起来。**如 I(7)，S(30)为合法，而 I7，I30 这二种写法不是数组元素名，而是简单变量名。

**（三）下标可以是常数、变量或表达式。**如 A(8)，A(I)，A(I+3)为合法，其中 I 应事先被赋值。如果 I 的值为 3，则 A(I)表示 A(3)，A(I+3)表示 A(6)。

下标又可以是下标变量，如 A(A(3))，如果 A(3)的值为 5，则 A(A(3))就是 A(5)。

**（四）下标值应大于或等于零，不能为负值。**如 A（0），A（100）为合法，而 A（—8），A（—100）为非法。

如果下标的值不是整数，如 A（3. 5），则被自动取整为 A（3）。

**（五）在计算机中，一个数组占据内存中一片连续的存储单元。**各数组元素在内存中是顺序存储的。例如，I（1）在最前面，I（2）紧接之，然后是 I（3），I（4）……等。如下表所示。

| I(1) | I(2) | I(3) | I(4) | I(5) | I(6) | I(7) | I(8) | I(9) | I(10) |
|---|---|---|---|---|---|---|---|---|---|
| 1.01 | 0.99 | 0.98 | 1.00 | 1.02 | 1.00 | 0.97 | 1.00 | 1.02 | 0.99 |

所以说，数组很自然地反映了计算机内大批数据存放的特点。数组与循环语句结合使用，可以很方便地处理一批数据。如：

```
10 FOR I=1 TO 100
20    INPUT A (I)
30 NEXT I
```

可以用它连续输入 100 个数据，存放在 A（1）到 A（100）中。如果不用数组而用简单变量来编程序，显然是很麻烦的。

## §6.2 一 维 数 组

如果一个数组，它的元素只有一个下标，那么这个数组称为一维数组，或者说用一个数组名和一个下标就能唯一地标识一个数组元素。例如，S 是学生成绩数组，S（1），S（2），……S（30）代表第 1，2，…30 个学生的成绩。用一个下标就确定了数组元素在数组中的位置。S 是一维数组。一维数组中各元素又称“单下标变量”。

请看下面简单的例子：

**【例 6.1】**

```
10   FOR I=1 TO 3
20     A (I) =I
30     PRINT A (I),
40   NEXT I
50   END
```

在第一次循环中，I=1，在执行 20 语句时，将 1 赋给 A（1），然后打印出 A（1）的值 1。第二次循环中，I=2，将 2 赋给 A（2），打印 A（2）的值 2。第三次循环中，I=3，将 3 赋给 A（3），打印 A（3）的值 3。这个程序运行后打印结果为：

```
RUN
 1              2              3
```

注意在使用数组元素时，如果下标是变量，请注意，实际起作用的是变量的值而不是变量的名。如下面二个程序段作用完全相同。

```
(a) 10 FOR  I=1 TO 10
    20  S (I) =I
    30 NEXT I
```

```
(b) 10 FOR  J=1 TO 10
    20  S (J) =J
    30 NEXT J
```

二者作用都是把 1，2，……10 这十个数赋给数组元素 S（1）到 S（10）。如果在上面（a）程序段后面加以下语句：

```
40  FOR J=1 TO 10
50    PRINT S (J),
60  NEXT j
```

执行 10～60 语句时将打印出 1，2，……10 十个数。

**【例 6.2】** 输入 10 个学生的成绩，打印出其平均值。

```
10  SUM=0                  ┐
20  FOR I=1 TO 10          │
30    INPUT S (I)          ├读入 10 个值赋给 S 数组的十个元素。
40    SUM=SUM+S (I)        │
50  NEXT I                 ┘
60  AVER=SUM/10            ┐
70  PRINT "Arerage="; aver ┘ 计算并打印平均值
80 END
```

**【例 6.3】** 一质点作自由落体运动，每次测其落地的时间 t，然后求此质点的离地高度 s。共测 n 次（设 n<10）。

$$s = \frac{1}{2}gt^2 \qquad g = 9.80$$

```
10   G=9.8
20   INPUT " N="; N          （输入 N 值）
30   FOR I=1 TO N                  ┐
40     PRINT " T ( "R; I;") =";    ├输入 N 个 t 的值
50     INPUT T(I)                  │
60   NEXT I                        ┘
70   FOR I=1 TO N          ┐
80     SUM=SUM+T(I)        ├计算平均时间
90   NEXT I                │
100  T1=SUM/N              ┘
110  S=G * T1 * T1/2
120  PRINT " CALCULATED DISTANCE="; S（110 和 120 语句为计算和打印距离）
130  END
```

程序运行结果为：

```
RUN
N=? 5↙
T (1) =? 8.95↙
T (2) =? 8.9↙
T (3) =? 9.01↙
T (4) =? 8.99↙
```

```
T (5) =? 9.11↙
CALCULATED DISTANCE= 396.1947
```

为了使读者清楚地了解每一步的作用，在程序中设了两个循环，第一个循环的作用是输入 $n$ 个时间 $t$ 的值，第二个循环的作用是把它们的值加起来。当然程序可以简化，将两个循环合并成一个循环。即将 30 到 90 语句写成：

```
30 FOR I=1 TO n
40  PRINT "t ("; i;") =";
50  INPUT t (i)
60  SUM=SUM+t (i)
90 NEXT I
```

**【例 6.4】** 设我国每年人口增长率分别为 3%，2.5%，2%，1.5%，1.0%，0.5%，请计算在公元 2000 年、2050 年和 1000 年后我国人口总数以及每平方米人口密度。以 1982 年人口为基数（10 亿 3 千万人）。全国土地面积为 960 万平方公里。

```
20  P1=1.03E+9
30  READ Y(1),Y(2),Y(3)
40  PRINT "RATE","2000","2050","AFTER 1000 YEARS"
45  PRINT
50  FOR R=0.03 TO 0.005 STEP -0.005
60    FOR I=1 TO 3
70    P(I)=P1 * (1+R) ∧ Y(I)
75      A(I)=P(I)/960E+10
80  NEXT i
90  PRINT R, P (1), P (2), P (3)
100  PRINT "  ", A (1), A (2), A (3)
110  NEXT R
140  DATA 18, 68, 1000
200  END
RUN
```

| RATE | 2000 | 2050 | AFTER 1000YEARS | |
|---|---|---|---|---|
| 0.03 | 1.7535E+09 | 7.6872E+09 | 7.08046E+21 | (人口) |
| | 1. 82656E-04 | 8.0075E-04 | 7.37547E+08 | (密度) |
| 0. 025 | 1. 60644E+09 | 5.52153E+09 | 5.45384E+19 | (人口) |
| | 1.67337E-04 | 0.75159E-04 | 5.68108E+06 | (密度) |
| 0.02 | 1.47109E+06 | 3.95957E+09 | 4.10212E+17 | (人口) |
| | 1.53238E-04 | 2.95294E-04 | 42730.4 | (密度) |
| 0.015 | 1.34656E+09 | 2.83483E+09 | 3.01216E+15 | (人口) |
| | 1.40266E-04 | 2.95294E-04 | 313.766 | (密度) |
| 0.01 | 1.23203E+09 | 2.02623E+09 | 2.15879E+13 | (人口) |
| | 1.28336E-04 | 2.11065E-04 | 2.24873 | (密度) |
| 5E-03 | 1.12674E+09 | 1.44587E+09 | 1.50972E+11 | (人口) |
| | 1.17368E-04 | 1.50611E-04 | 0.015726 | (密度) |

因为要求计算 2000 年，2050 年和 1000 年后的人口和密度，因此设三个数组 Y、P 和 A。Y 数组中存放从 82 年开始的年数，P 数组中存放上述年份时的人口，A 数组存放相应年份的人口密度（人/米$^2$）。

P1 为 82 年人口（10 亿 3 千万）。读 Y（1）＝18（因 2000－1982＝18），Y（2）＝68，Y（3）＝1000。用双层循环，外循环是控制增长率 R，第一次 R 循环计算 R＝0.03 时的情况。最后一个循环处理 R＝0.005 的情况。内循环（I 循环）计算在 2000 年，2050 年时和 1000 年后的人口和密度。在执行第一次 I 循环时，I＝1，70 语句相当于：

70 P(1)＝P1 * (1＋R) ∧ Y(1)

而 R＝0.03，Y(1)＝18。计算出 P(1)＝1.7535E＋09。75 语句相当于：

75 A(1)＝P(1)/960E＋10

因为 960 万平方千米＝$960\times10^{10}$ 平方米。故 2000 年时每平方米上人口为 P(1)/960E＋10，计算出 A(1)＝1.82656E－04。在执行完整个的循环(三次)后，打印出 R＝0.03 时的各种所需数据。90 语句打印人口，100 语句打印人口密度。然后再进行 R＝0.025 时的计算。

从运行结果可知：当 R＝0.03 时，公元 2000 年时，人口将达 17 亿，1000 年后人口将达 $7.08\times10^{21}$，即 7 万亿亿人。那时每平方米上将居住 $7.37\times10^{8}$ 人，即 7 亿人都住在一平方米上。即使全中国（包括沙漠和高山）都盖了摩天大楼也住不下这么多人啊！而现在我国每平方米上只有万分之一人。多么惊人的对比！我们可不能使我们的子孙将来在中国无立锥之地哟！

## §6.3 二 维 数 组

一维数组有时不能满足要求，例如，要统计一个班的成绩，有三个学生，五门课程，可以用一个二维数组 S 表示：

$$S=\begin{bmatrix}78 & 67 & 75 & 87 & 71\\ 56 & 71 & 88 & 91 & 50\\ 99 & 79 & 81 & \boxed{89} & 98\end{bmatrix}$$

其中以一行代表某一个学生的成绩，一列代表某一门课的成绩。若要表示第三个学生第四门课的成绩，可写为：

S（3，4）

↙　↘

学生号　课程号

它代表第三行第四列的数据，就是“89”。可以看到为了表示某一学生某一门课成绩，必须用两个下标，现在我们用第一个下标代表学生号，用第二个下标代表课程号。在二维数组中，第一个下标称为“行下标”第二个下标称为“列下标”。在二维数组中，必须用两个下标方能唯一地确定一个数组元素在数组中的位置。二维数组的元素又称“双下标变量”。

**【例 6.5】** 计算全班学生的每门课的总平均成绩。今设全班有六个学生，每个学生考 5 门课，以 S 数组放这些数据。以第一个下标表示学生号，第二个下标表示课程号。S（I，J）表示第 I 个学生的第 J 门课的成绩。平均成绩取小数点后一位，第二位四舍五入。

程序如下：

```
10   FOR I=1 TO 6
20     FOR J=1 TO 5
30       READ S (I, J)  } 读一个学生的成绩  } 读入六个学生成绩
40     NEXT J
50   NEXT I
60   PRINT "COURSE", "AVERAGE" (打印表头)
70   FOR J=1 TO 5
80     T=0
90     FOR I=1 TO 6
100       T=T+S (I, J)  } 把六个学生第一门课成绩相比
110     NEXT I
120     A=INT (T/6*10+0.5) /10 (求一门课平均成绩)
130     PRINT J, A
140   NEXT J
150   DATA 86.5 87,91,95,68.5
155   DATA 83,77,38,90,67
160   DATA 80 90 91 88,85
165   DATA 99,100,90 85 97
170   DATA 88,85,79,96,86
180   DATA 85,88,54,86,83
200   END

RUN
COURSE                          AVERAGE
  1                               86.9
  2                               87.8
  3                               73.8
  4                               90
  5                               81.1
```

**请考虑：如果需要统计每个学生五门课的平均成绩，程序应如何修改？10～50语句需不需要修改？**

**【例6.6】** 解二元一次联立方程组

$$\begin{cases} 4X_1 - 3X_2 = 6 \\ -2X_1 + 4X_2 = -5 \end{cases}$$

**其一般形式为：**

$$\begin{cases} a_{11}X_1 + a_{12}X_2 = C_1 \\ a_{21}X_1 + a_{22}X_2 = C_2 \end{cases}$$

$$X_1 = \frac{c_1a_{22} - a_{12}c_2}{a_{11}a_{22} - a_{12}a_{21}} \qquad X_2 = \frac{a_{11}c_2 - a_{21}c_1}{a_{11}a_{22} - a_{12}a_{21}}$$

**程序为：**

```
10 FOR I=1 TO 2                              ⎫
20   FOR J=1 TO 2                            ⎪
30     PRINT "A (";I;",";J;") =";            ⎬输入系数
40     INPUT A (I, J)                        ⎪
50   NEXT J                                  ⎭
60  NEXT I
80  INPUT "C (1), C (2) =";C (1), C (2)          (输入常数C1, C2)
90  D=A (1, 1) *A (2, 2) -A (1, 2) *A (2, 1)
100  X1=C (1) *A (2, 2) -C (2) *A (1, 2)
110  X1=X1/D
120  X2=C (2) *A (1, 1) -C (1) *A (2, 1)
130  X2=X2/D
140  PRINT "X1=", X1, "X2="; X2
150  END
```

程序运行情况如下：

```
RUN
A (1, 1) =? 4↙                   ⎫
A (1, 2) =? -3↙                  ⎪
A (2, 1) =? -2↙                  ⎬方程的系数和常数由键盘输入
A (2, 2) =? 4↙                   ⎪
C (1), C (2) =? 6, -5↙           ⎭
X1= 0.9  ⎫
X2=-0.8  ⎭打印出二个根
```

## §6.4 数组说明语句（DIM 语句）

在使用数组时，需要考虑两个问题：(1) 数组是几维的？一维？二维？或三维？(2) 每一维有多少元素。

应当将以上两个信息告诉计算机，以便开辟足够的内存单元来存贮数据。在 BASIC 中用 DIM 语句来完成这一工作（DIM 是 DIMension 的缩写），这叫做"定义"（或"说明"）一个数组。如果写：

```
5 DIM A (100), B (20, 30)
```

语句中的 A (100) 和 B (20, 30) 称作数组说明符。其作用是对指定的数组 A 和 B 作说明。请注意，DIM 语句中的 A (100) 并不是指 A 数组中的第 100 个元素，而是指 A 数组的大小，即指的是：该一维数组的下标值最大可以用到 100（从 A (0) 到 A (100)，共 101 个元素）。B 数组的第一维的下标值最大可以用到 20（从 0 到 20），第二维的下标值最大可用到 30（从 0 到 30）即定义的这个二维数组 B 共有 21 行 31 列，有 21×31=651 个元素。

如果在一个程序中用了上面的 DIM 语句定义了数组 A 和 B，则在此程序中使用数组元素时，下标值不应超过上述范围。例如下面的用法是正确的：

```
80  T=(A(55)+A(10))/(B(18,19)*B(0,30))
```

而下面的用法是不正确的：

```
80 T=(A(102)+A(-3)/(B(0,41)*B(25,35)
```

因为这些数组元素的下标值已超过了定义数组时所指定的下标最大值。

关于 DIM 语句，有几点说明：

(1) 应该把定义数组的数组说明符和数组元素这二者区分开来。前者只出现在 DIM 语句中。请区别下面两个语句中的 A(100) 的含义：

```
5 DIM A(100)
 ⋮
 ⋮
100 T=A(100)
```

5 语句是定义数组的大小，说明它包含 101 个元素，而 100 语句中的 A(100) 是引用数组中下标为 100 的元素 A(100) 的值。

(2) 下标值是从 0 开始算的，如果有：

```
10 DIM A (50)
```

则 A 数组中共有 51 元素。如果有：

```
10 DIM B(N, M)
```

则 B 数组第一维有 (N+1) 个元素，第二维有 (M+1) 个元素，数组元素总数为 (N+1) × (M+1)。下标值“从 0 开始”这一点常被忽略。如果已定义一个二维数组：

```
10 DIM A (2, 3)
```

则它代表以下一个矩阵：

$$\begin{bmatrix} A(0,0) & A(0,1) & A(0,2) & A(0,3) \\ A(1,0) & A(1,1) & A(1,2) & A(1,3) \\ A(2,0) & A(2,1) & A(2,2) & A(2,3) \end{bmatrix}$$

(3)二维数组中各元素在内存中排列顺序为：按行存放，即先放第一行，然后第二行，……。即：

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| A(0,0) | A(0,1) | A(0,2) | A(0,3) | A(1,0) | A(1,1) | A(1,2) | A(1,3) | A(2,0) | A(2,1) | A(2,2) | A(2,3) |

(4) 人们往往不习惯使用 0 下标，而希望下标值从 1 开始，MS BASIC 允许程序设计者根据需要选择下标的下界为 0 或 1。使用 OPTION BASE 语句：

```
OPTION BASE n
```

n 可以是 0 或 1 二者之一，如果希望下标下界从 1 开始，则应在 DIM 语句之前，先写：

```
10 OPTION BASE 1
20 DIM A(2, 3)
```

则二维数组 A 只包含 6 个元素：

$$\begin{pmatrix} A(1,1) & A(1,2) & A(1,3) \\ A(2,1) & A(2,2) & A(2,3) \end{pmatrix}$$

如果不写 OPTION BASE 语句，则默认为 OPTION BASE 0，即下标下界为 0。

(5) 允许在 DIM 语句中用变量来说明各维元素的个数。但应在 DIM 语句出现前使这些变

量具有确定的值。如：

```
10  INPUT N, M
20  DIM B (N, M)
```

这就可以在每次运行时使N，M具有临时输入的值以改变B数组的大小，使程序适应不同的用途。

(6) 在同一个DIM语句中可以说明一个或若干个数组，它们可以是维数不同的数组，如：

```
10  DIM A(50),B(10,15),C(100,2,10)
```

但不同的数组不能用同一个数组名。如：

```
20  DIM D(10),D(20,20)
```

是错误的。因为用同一个D既代表一个一维数组又代表一个二维数组是不行的。

(7) 如果程序中不写DIM语句，则系统自动分配该数组每一维包括11个元素。如果程序中用到A (5)，B (5，3) 数组元素，而没有用过DIM语句说明A，B的维数和大小。系统隐含已定义了以下的语句：

```
10  DIM A (10), B (10, 10)
```

在§6.2和§6.3中的例题中，由于所用的数组元素下标值都没有超过10，因此没有用DIM语句。

因此，如果程序中用的数组每维不超过11个元素，可以不必写DIM语句。但为了节省内存和避免使用中不必要的失误，建议养成根据数组的实际大小来定义数组的习惯。即使每维不超过11个元素，也用DIM语句，如：

```
10  DIM T(3,3),B(8)
```

注意在定义数组时不要不加考虑地定得过大。例如只用到50个元素，却定义：

```
10  DIM A(100)
```

虽是合法的，但不必要，浪费内存。数组定大了，占内存空间很大，例如：

```
10  DIM N(100,100)
```

N数组共有101×101＝10201个元素，每个元素在内存中占四个字节，N数组就占了四万个字节，约40K。而有的微型机的内存才只有64K。因此在定义数组时必须要十分注意节约使用内存。许多程序在微型机上不能运行，就是由于数组太大而内存不够之故。节约内存的关键在于节约使用数组。

(8) MS BASIC允许定义数组的维数最大为255(即255维)，每一维最多可以有32767个元素。(但有些BASIC只允许使用三维以下的数组)。

## §6.5 程序举例

**【例6.7】** 输入若干个学生的学号和成绩，要求打印出高于平均分的学生学号和成绩。

可以设计出以下算法：

S1：输入$n$个学生的学号和成绩

S2：求平均成绩

S3：将 $n$ 个成绩与平均分比较，打印出高于平均分的学生学号和成绩。

由于 $n$ 个学生的分数需要保存，以便在 S2 步骤中与平均分比较，因此必须使用数组。

```
10   INPUT "Enter the number of students:", N
20   DIM S(N),P(N)
30   FOR I=1 TO N          }
40     INPUT S(I),P(I)     } 输入 n 个学生学号和成绩
50   NEXT I                }
60   TOTAL=0               }
70   FOR I=1 TO N          }
80     TOTAL=TOTAL+P(I)    } 计算平均分数
90   NEXT I                }
100  AVER=TOTAL/N          }
105  PRINT "aver=";AVER
110  FOR I=1 TO N
120    IF P(I)>AVER THEN PRINT S(I),P(I)
130  NEXT I
140  END
RUN
Enter the number of students: 5↙
? 101, 100↙
? 102, 90↙
? 103, 98↙
? 104 78↙
? 105, 88↙
aver=90. 8
101          100
103          98
```

程序中以 S（I）代表第 I 个学生字号，P（I）代表第 I 个学生的分数。

**【例 6.8】** 输入 10 个数，要求按由小到大次序打印出来。

算法如下：

S1：输入 10 个数给 X 数组

S2：对 10 个数排序

S3：输出已排序的 X 数组

对 S2 还要细化、关键是如何实现排序。排序的方法很多。我们现在用“比较交换法”。基本思路如下，设有 10 个数，分别为 X（1）……X（10），值如下所示。

| 10 | 15 | −3 | 0 | 7 | 9 | −15 | 31 | 44 | 5 |
|---|---|---|---|---|---|---|---|---|---|
| X(1) | X(2) | X(3) | X(4) | X(5) | X(6) | X(7) | X(8) | X(9) | X(10) |

先将 X（1）与 X（2）相比，如果 X（2）<X（1）则将 X（1）与 X（2）互换。此时，X（1）的值是 X（1）与 X（2）之中较小的那个。再将 X（1）与 X（3）比较，处理方法同上。

然后 X（1）与 X（4）～X（10）各数比较，每次都如法泡制。比完这一轮后，10 个数中最小的数已在 X（1）中了。在第二轮中，将 X（2）与 X（3）～X（10）比较，处理方法同上。比完第二轮后，10 个数中第二小的数在 X（2）中。……如此共比较 9 轮，在第 9 轮中，将 X（9）与 X（10）比较。请读者按上述步骤对以上 10 个数进行处理，看能否达到目的。S2 可细化为：

S2：当 i=1 到 9 时进行以下操作：

将 X(i)到 X(10)中最小的数放到 X(i)中可以画出 N－S 图．见 6.1。

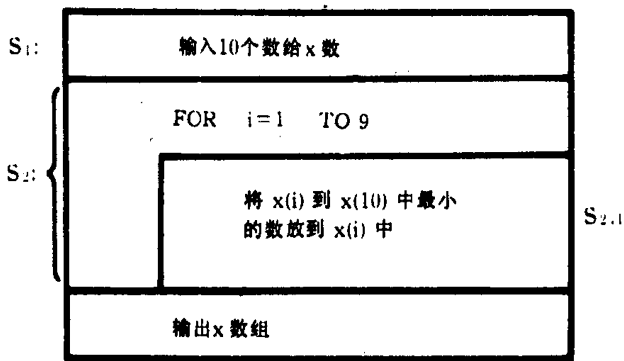



图 6.1

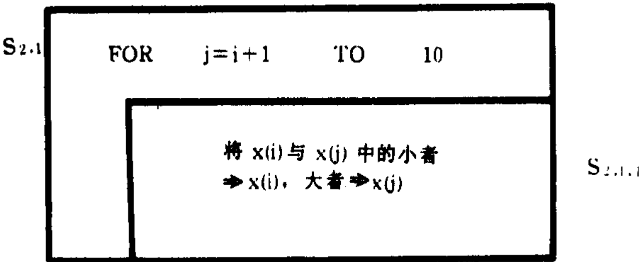



图 6.2

对 S2.1 又可以细化为：

S2.1：将 X（i）与 X（i+1）到 X（10）各元素逐一比较，每次比较中，将小数⇒X（i），大数放到另一元素中，画出 S2.1 的 N－S 图（见图 6.2）。

S2.1.1 又可以细化为图 6.3。

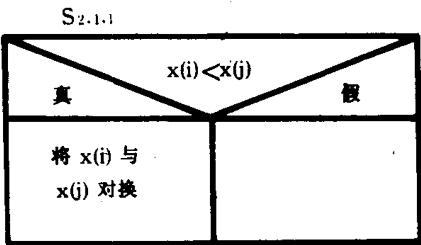



图 6.3

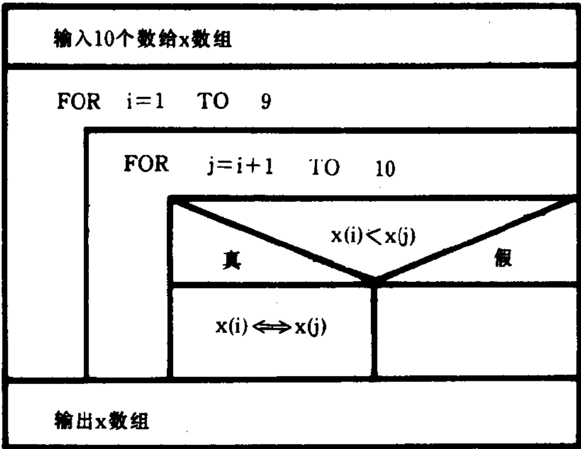



图 6.4

将以上各图合并为图 6.4。　　根据流程图编写出程序：

```
10  DIM X (10)
20  FOR I=1 TO 10
30    INPUT X (I)
40  NEXT I
```

```
50  FOR I=1 TO 9
60    FOR J=I+1 TO 10
70      IF X (I) >X (J) THEN T=X (I): X (I) =X (J): X (J) =T
80    NEXT J
90  NEXT I
100  FOR I=1 TO 10
110      PRINT X (I);
120  NEXT I
130  PRINT
140  END
```

在运行开始后按以下顺序输入 10 个数，每输入一个数后，按回车，数据如下：

10，15，—3，0，7，9，—15，31，44，5

输出以下结果：

—15 —3　0　5　7　9　10　15　31　44

这种排序方法容易理解，但交换次数多，只要 $X(I) > X(J)$，就要交换一次数据，而交换一次数据要执行三个赋值语句，费时多。如果原来的数据恰 好是由大到小排列的，则每比一次就要交换一次，共需比较 $\sum_{n=1}^{9} n = 45$ 次，交换 45 次，执行 $45 \times 3 = 135$ 个赋值语句。可以对此算法作改进，在每一轮比较中不必每当 $X(I) < X(J)$ 就交换数据，而在 $X(I)$ 与 $X(I+1)$ 到 $X(10)$ 都比完后，将 $X(I)$ 与 $X(I+1) \sim X(10)$ 中值最小的那个元素对换即可。在每轮的各次比较中，将值最小的元素的下标放在变量 $L$ 中，比完一轮后，将 $X(I)$ 与 $X(L)$ 对换即可。这种排序方法称为“选择排序法”。流程图见图 6.5。

```
10 FOR I=1 TO 10 ┐
20 READ X (I)    ├ 读入十个数，赋给 X 数组
30 NEXT I        ┘
40  FOR I=1 TO 9
45    L=I
50    FOR J=I+1 TO 10
60      IF X(J)<X(L) THEN L=J
70    NEXT J
80    IF L<>I THEN T=X (I): X (I)=X(L):
      X(L)=T
90  NEXT I
100  FOR I=1 TO 10
110    PRINT X (I)
120  NEXT I
130  DATA 10,15,-3,0,7,9,-15,31,44,5
140  END
```

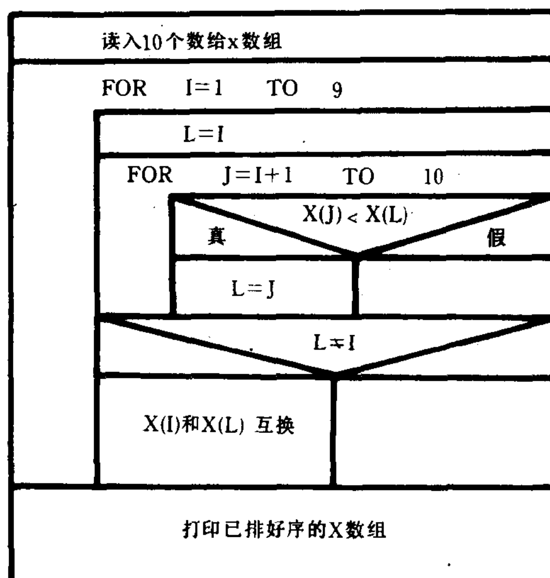



图 6.5

80 语句的作用是：当比完一轮后若得到的最小值不是 X (I) 本身时才将 X (I) 与 X (L) 交换。如果最小值就是 X (I)，此时 L=I，不进行交换。

本程序用 READ 语句从 DATA 语句中读入 10 个数，可以看到：“选择法”与“比较交换法”相比，比较次数一样，但交换次数大大减少了。最多交换 9 次（每比完一轮交换一次，即

执行一次外循环交换一次)。执行时间大大减少。

**【例 6.9】** 某学校对如何实行改革拟出三个方案，用民意测验的办法征求意见。需要统计每一方案所得票数和分别得自教员、学生、职工这三部分人的票数，票的格式见图 6.6。

参加民意测验的人在(一)和(二)项中的 1，2 或 3 个数字上任选定一个并画一个圈。然后将画圈的数字打入计算机由计算机处理。

今设一个二维数组 C，第一维代表“方案”、第二维代表“部门”。用 C(1，2)代表投票赞成方案 I 的学生的人数，如果有一张选票在①，②上画图，就表示有支持方案 I 的是学生。此时应输入“1，2”，然后使 C(1，2)的值加 1。假设数据用 DATA 语句存放，以“－1，－1”为终止标志。流程图可以用 6.7 表示。

**征求意见表**

(一) 你赞成哪一种方案

1. 方案 I
2. 方案 II
3. 方案 III

(二) 你属于哪一部门

1. 教员
2. 学生
3. 职工

图 6.6

在未“开票”时，使 C 数组中每个元素都为初值 0。读入一对数据，就使相应的数组元素值加 1。直到读入“－1”为止，程序为：

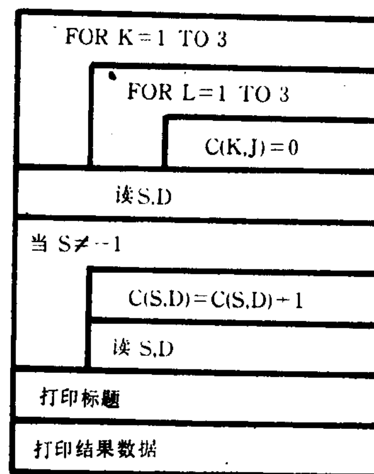



图 6.7

```
10  REM VOTE COUNTING
20  DIM C (3, 3)
30  FOR K=1 TO 3
40    FOR L=1 TO 3
50       C (K, L) =0
60    NEXT L
70  NEXT K
80  READ S, D
90  WHILE S<>-1
100   C (S, D) =C (S, D) +1
105   READ S, D
110  WEND
120  PRINT "SCHEMA", TAB (12); "TEACHER"; TAB (24); "STUDENT"; TAB (36); "WORKER";
     TAB (48); "TOTAL"
130  PRINT
140  PRINT "SCHEMA 1"; TAB (12); C (1, 1); TAB (24); C (1, 2); TAB (36); C (1, 3); TAB
     (48); C (1, 1) +C (1, 2) +C (1, 3)
150  PRINT "SCHEMA 2"; TAB (12); C (2, 1); TAB (24); C (2, 2); TAB (36); C (2, 3); TAB
     (48); C (2, 1) +C (2, 2) +C (2, 3)
160  PRINT "SCHEMA 3"; TAB (12); C (3, 1); TAB (24); C (3, 2); TAB (36); C (3, 3) TAB
     (48); C (3, 1) +C (3, 2) +C (3, 3)
170  DATA 3, 1, 2, 2, 3, 2, 1, 2, 1, 1, 2, 1, 3,
```

```
180  DATA 2, 2, 1, 1, 1, 3, 3, 1, 3, 2, 2, 2, 1, 2
190  DATA 2, 1, 1, 3, 2, 3, 3, 2, 1, 1, 3, 3, 1, 2
200  DATA 2, 2, 2, 3, 3, 3, 1, 3, 1, 2, 1, 1, 2, 2, -1, -1
210  END
RUN
```

| SCHEMA<br>（方案） | TEACHER<br>（教员） | STUDENT<br>（学生） | WORKER<br>（职工） | TOTAL<br>（总计） |
|---|---|---|---|---|
| SCHEMA 1 | 4 | 4 | 3 | 11 |
| SCHEMA 2 | 2 | 5 | 2 | 9 |
| SCHEMA 3 | 2 | 3 | 3 | 8 |

从打印结果可以看到每一方案的总票数和来自教员、学生、职工的票数。请考虑为什么200语句中最后要用两个“－1”而不是一个“－1”作“终止标志”。请仔细分析90语句的作用。

120～160语句中用了TAB函数，TAB函数是打印位置函数，TAB（X）的作用是把当前输出位置移至本行第X例，（注意X的值在1～225之间）。例如若A＝10，B＝－20，则

10 PRINT TAB（12）；A；TAB（20）；B

打印（或显示）结果如下：

| 列数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 输出 | | | | | | | | | | ␣ | 1 | 0 | | | | | | | | － | 2 | 0 |

A的值从第10列开始输出，由于A是无符号整数，它前面应有一个空格（符号位），即将“␣ 1 0”输出在第10～12列：而B是负数，负号占一列，因此“－20”输出在第20～22列上。

TAB是Tabulate（制表）的缩写。本例用TAB函数控制输出数据的位置。注意TAB（x）后面用分号与其后的输出量相分隔。

**【例6.10】** 打印相辉三角形。

相辉三角形的每一行是 $(x+y)^n$ 的展开式的各项的系数。如第1行是 $(x+y)^0$ 的系数，第2行的“1，2，1”是 $(x+y)^2$ 展开式 $x^2+2xy+y^2$ 各项的系数。要求输出结果如下：

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
1   7   21  35  35  21  7   1
1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84  36  9     1
```

可以找出以下规律，对角线和每行的第一列都为1，其余各项是它的上一行中前一个元素和上

一行的同一列上的元素之和。例如，第 4 行第 2 列的值为 3，它是第三行上第 2 列和第 3 列元素值之和，可用下式表示：

$$A(i,j) = A(i-1,j-1) + A(i-1,j)$$

程序如下：

```
10   INPUT " N="; N
20   DIM A (N, N)
30   FOR I=1 TO N
40     A (I, I) =1
50     A (I, 1) =1
60   NEXT I
70   FOR I=3 TO N
80     FOR J=2 TO I-1
90       A (I, J) =A (I-1, J-1) +A (I-1, J)
100     NEXT J
110   NEXT I
120   FOR K=1 TO N
130     FOR J=1 TO K
140       PRINT TAB (5*J); A (K, J);
150     NEXT J
160     PRINT
170   NEXT K
180   END
```

30～60 语句是将各行第 1 列和对角线元素置 1，因此第 2 行（只有两个元素）都已有值了。所以只需从第 3 行开始计算各元素的值即可。故 70 语句循环变量 $I$ 的值从 3 开始变到 10。

120～160 语句输出相辉三角形，请注意 130 语句循环变量的终值为 $K$ 。这样使第 1 行输出 1 个数，第 2 输出印 2 个数，第 K 行输出 K 个数。140 语句的作用是使每个被输出的元素占 5 列的宽度。例如，当 $K=3$ 时，输出第 3 行数据，内循环变量 $J$ 由 1 变到 3，内循环的作用是输出 K 个数据，第 1 个数从第 5 列开始输出，第 2 个数从第 10 列开始输出，第 3 个数从第 15 列开始输出。其余行情况类似。请注意 160 语句的作用；在执行完内循环输出完一行中数据后使之换行。如无此 160 语句，则各行数据会连续显示在一行上。

除了可以利用 TAB 函数控制所输出数据的位置外，还可以利用 SPC 函数（空格函数），它的作用是使输出位置向右移动若干列。例如：

```
10 PRINT "A"; SPC (3); "B"; SPC (2); "C"
```

输出为：

```
A ␣␣␣B␣␣C
```

**【例 6.11】** 有一个 $n \times m$ 的矩阵，找出其中值最大的那个元素的值以及其所在的行号和列号。

算法如下：

1. 输入 $n$、$m$ 和 $n \times m$ 矩阵中各元素的值；
2. 先将 A（1，1）的值赋给 MAX

3. 将 MAX 与各元素值相比，如果有某一个 A (I, J) >MAX，则将此 A (I, J) ⇒MAX，同时将此时的 I，J 值记下来赋给变量 ROW（行号）和 COLUM（列号）。

4. 最后输出 MAX 和 I，J 的值。

算法见图 6.8。

```
10  INPUT " N, M="; N, M
20  DIM A (N, M)
30  FOR I=1 TO N
40    FOR J=1 TO M
50      INPUT A (I, J)
60    NEXT J
70  NEXT I
80  MAX=A (1, 1)
90  FOR I=1 TO N
100  FOR J=1 TO M
110    IF A (I, J) >MAX THEN
       MAX=A (I, J): ROW=I: COLUM=J
120  NEXT J
130  NEXT I
140  PRINT " THE LARGEST NUMBER IS A (";
     ROW;",";COLUM;") ="; MAX
150  END
```

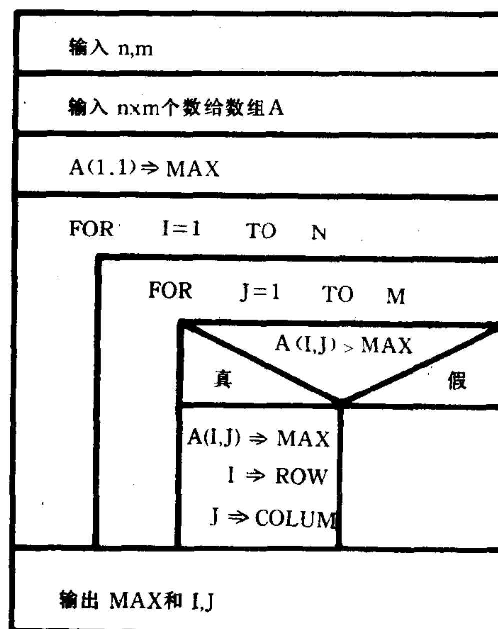



图 6.8

如果想处理的矩阵为：

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 9 & 8 & 7 & 6 \\ 0 & 1 & 8 & 7 \end{pmatrix}$$

在运行时输入情况如下：

N, M=3, 4↙

然后按以下顺序输入数据：1，2，3，4，9，8，7，6，0，1，8，7（每输入一个数据按一次回车）。程序输出：

THE LARGEST NUMBER IS A(2,1)= 9

## 习　　题

6.1　请写出 A 数组中各元素的值

(1)
```
10 DIM A (5)
20 FOR I=1 TO 5
30   A (I) =I*I
40   A (I) =A (I) *I
50 NEXT I
```

(2)
```
10 DIM A (5, 5)
20 FOR I-1 TO 5
30     FOR j=1 TO 5
```

```
40        A (I, J) =0
50     NEXT J
60     A (I, I) =1
70 NEXT I
```

(3)
```
10 DIM A (5, 5)
20 FOR I=1 TO 5
30     IF I=1 OR I=5 THEN FOR J=1 TO 5: A (I, J) =0: NEXT J: GOTO 70
40     FOR J=1 TO 5
50         IF J=1 OR J=5 THEN A (I, J) =0 ELSE A (I, J) =1
60     NEXT J
70 NEXT I
```

6.2 输入 10 个数给 X 数组，找出其中值为最大的元素并将其与第一个元素互换，找出值最小的元素并将其与最后一个元素互换，其它元素不动。如：

原来：8，7，9，15，0，3，−8，19，31，5

输出：31，7，9，15，0，3，5，19，8，−8

6.3 有一个一维数组 X，含 $n$ 个元素，要求对 X (1) 与 X (n) 对换，X (2) 与 X (n−1) 对换，……如：

原来： 1，13，15，27，9，21，33

对换后：33，21，9，27，15，13，1

(注意：n 可以是奇数或偶数)

6.4 已有一个已排好序的数组，今输入一个数，要求插入到数组中，插入后仍按原规律有序。

6.5 求 Fibonacci 数列前 50 个数，把它们放到 F 数组中的 F (1) 到 F (50) 中，并打印出来。

6.6 输入一个 3×3 的二维数组 A，要求将其行与列互换，得 B 数组，并输出数组 A 和 B (各分别按三行输出)

$$A=\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}\qquad B=\begin{pmatrix}1 & 4 & 7\\2 & 5 & 8\\3 & 6 & 9\end{pmatrix}$$

6.7 有一个 $n\times m$ 的二维数组，求它的最靠外边的行和列各元素的值之和。

6.8 有一 $n\times n$ 数组，求两条对角线元素之和。

6.9 有二个 $n\times m$ 的数组，要求将它们相同位置的元素相加 (矩阵加法)

6.10 将例 6.5 改为求每个学生各门课平均成绩，取小数点后二位，第三位四舍五入。

6.11 假如有四个商店，它们当日卖出的商品 A、商品 B 和商品 C 的数量如下表所示。

| 商店名 \ 售出数量 \ 商品名 | 商品 A | 商品 B | 商品 C |
|---|---|---|---|
| 第一商店 | 56 | 64 | 83 |
| 第二商店 | 126 | 37 | 65 |
| 第三商店 | 81 | 194 | 112 |
| 第四商店 | 206 | 120 | 191 |

各商品的价格如下：

| 商　　品 | 价格(元) |
|---|---|
| 商品 A | 12.5 |
| 商品 B | 6.8 |
| 商品 C | 4.3 |

请编程序，分别求出各商店当日的总营业额。

6.12　按下面形式打印相辉三角形：

```
        1
       1 1
      1 2 1
     1 3 3 1
    1 4 6 4 1
        ⋮
```

6.13　打印魔方阵。所谓魔方阵是指这样的方阵：它的每一行、每一列和对角线之和均相等。要求打印由自然数 1 到 $n^2$ 构成的魔方阵（n 为奇数），例如，当 $n=3$ 时，魔方阵为：

```
8  1  6
3  5  7
4  9  2
```

魔方阵中各数排列规律为：

①将“1”放在第 1 行当中一列；

②从“2”开始直到 $n\times n$ 止各数依次按下列规则放数：每一个数存放的行比前一个数的行数减 1，列数加 1；

③如果上一数的行数为 1，则下一个数的行数为 n（最下一行），如在 3×3 方阵中，1 在第 1 行，则 2 应放在第 3 行第 3 列。

④当上一个数的列数为 $n$ 时，下一个数的列数应为 1，行数减 1。如 2 在第 3 行第 3 列，3 应在第 2 行第 1 列。

请编程序，输入 $n$，打印出由 $n^2$ 个数组成的魔方阵。

6.14　找出一个二维数组中的“鞍点”，即该位置上的元素在该行上最大，在该列上最小，(也可能没有鞍点)，打印出有关信息。

# 第七章　函数与子程序

## §7.1　BASIC标准函数的分类及应用

在第二章§2.5中已简单介绍了BASIC的标准函数。BASIC把一些常用的运算事先做成子程序，在需要时调用。函数调用的形式为：

**函数名（自变量）**

一般的标准函数要求有一个自变量（如SIN（x），INT（a）），有的标准函数要求二个自变量（如LEFT$（A$，n））或三个自变量（如MID$（A$，n，m）），见附录Ⅱ。在引用函数时要用具体的值作为自变量，可以是常量、变量、或表达式。即：只能对确定的值求函数值。

每种BASIC系统提供的标准函数的种类、个数与功能不尽相同。功能强的BASIC提供丰富的标准函数，使用十分方便，使编写程度的工作量减少，程序简明易读。

一般BASIC包含以下几类标准函数：

（1）数学函数。这些函数的值是一个数值（如三角函数、对数函数、绝对值函数、平方根函数、随机函数等）。

（2）与字符串有关的函数，如函数LEN（x$）的作用是求字符串x$的长度。CHR$（n）的作用是得到一个ASCII码为n的字符。

（3）输入输出函数。例如有TAB函数（输出位置函数），SPC函数（空格函数）。INKEY$函数的作用是从键盘读一个字符。

（4）具有特定功能的专门函数。例如FRE函数可以查内存中自由空间字节数，VARPTR函数可以查变量在内存中地址等。

我们在前面几章的程序中运用标准函数来处理问题，例如用INT函数处理取$n$位小数和四舍五入以及判断整除等，用三角函数、EXP、SQR、LOG、ABS函数进行数值计算，用TAB函数控制打印位置等。应当说明，BASIC中一些数学函数的值是用近似计算公式求出的，例如：三角函数、指数函数（EXP函数）、对数函数（LOG函数）等都是用幂级数多项式形式的台劳公式进行计算的。如正弦函数sin（$x$）是用下面的台劳展开式计算的：

$$\sin x \approx \frac{x}{1}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\frac{x^9}{9!}+\cdots\cdots-\frac{x^{4n-1}}{(4n-1)!}+\frac{x^{4n+1}}{(4n+1)!}$$

因此，某些函数的求值不可避免地会出现微小的误差。

由于标准函数很多，无法一一介绍。读者需要时可查附录Ⅱ即可。使用较多的函数是数学函数和字符串函数。字符串函数将在第八章中介绍。在本节中只准备介绍一种使用较广泛的数学函数——随机函数的应用。随机函数在计算机游戏和计算机模拟中得到广泛的应用。

随机函数的一般形式为：

**RND [ (x)]**

方括弧内部分是任选的，即 RND，RND (x) 两种形式均合法。其作用是得到一个 (0，1) 间的随机数。这个随机数是由 BASIC 解释系统中一个随机数生成程序（又称“随机数生器”）产生的。随机数生成程序是根据一个复杂的公式来计算出一个随机数序列 (R1，R2，R3，……）的，每用一次 RND 函数就可得到这个随机数序列中的一个随机数。给这个“随机数生成程序”指定不同的初值（即随机数序列的“基因子”，或称“种子”），就可以得到不同的随机数序列。在 BASIC 中用 RANDOMIZE 语句设置随机数“种子”。其一般形式为

**RANDOMIZE [n]**

n 是一个整表达式，作为指定的“种子”，如果每次指定的 n 值相同，则得到相同的随机数序列。

(1) 不用 RANDOMIZE 语句，相当于按同一个“种子”进行计算，产生同一个随机数序列。

**【例7.1】**

```
10   FOR 1=1 TO 4
20     PRINT RND
30   NEXT 1
40   END
RUN
 .1213501
 .651861
 .8688611
 .7297625
RUN
 .1213501
 .651861
 .8688611
 .7297625
```

每次运行都得到同一个随机数序列。

(2)使用 RANDOMIZE 语句，在语句中可以不写 *n* 值，执行此语句时系统会询问 *n* 的值，可以临时从健盘输入。

**【例7.2】**

```
10   RANDOMIZE
20   FOR 1=1 TO 4
30     PRINT RND
40   NEXT 1
50   END
RUN
Random number seed (-32768 to 32767)? 2↙
 .8162134
 .4226789
 .9276842
```

```
.6269628
RUN
Random number seed (−32768 to 32767)? 3↙
 .2226007
 .5941419
 .2414202
 .2013798
RUN
Random number seed (−32768 to 32767)? 4↙
 .628988
 .765605
 .5551561
 .775797
RUN
Random number seed (−32768 to 32767)? 2↙
 .8162134
 .4226789
 .9276842
 .6269628
```

可以看到，当输入的 $n$ 值相同时，得到相同的随机数序列。

(3) 当用"RND (x)"形式时，x>0 时，每使用一次 RND (x) 可得到随机数序列中下一个随机数。用"RND"形式时（即省略 x），作用与此相同。

**【例7.3】**

(a)

```
10  FOR 1=1 TO 3
20    PRINT RND (1),
30  NEXT 1
40  END
```

(b)

```
10  PRINT RND (1),
20  PRINT RND (1),
30  PRINT RND (1)
40  END
```

二者运行结果相同：

```
. 1213501    .651861    .8688611
```

虽然每次都用 RND (1)，但得到的值不同，依次从随机数序列中找下一个随机数。

(4) 若 RND (x) 中 x 为零，即 RND (0)，得到与上一个随机数相同的随机数。

**【例7.4】**

```
10  FOR 1=1 TO 3
20    PRINT RND (0),
30  NEXT 1
40  END
RUN
.3116351    .3116351    .3116351
```

得到同一个随机数的值。

(5) 若 RND (x) 中 $x<0$，则重设"种子"而改变随机数序列，且不受 RANDOMIZE 语句影响。

【例7.5】

```
10   RANDOMIZE 5
20   FOR I=1 TO 3
30     PRINT RND (1),
40   NEXT I
45   PRINT
50   RANDOMIZE 5
55   X=RND (-1)
60   FOR I=1 TO 3
70     PRINT RND (1),
80   NEXT I
90   END
RUN
 3.537536E-02          .9370679          .8688921
 .6545178              .4623311          2.003121E-02
```

10 语句用 RANDOMIZE 指定了"种子"值 $n=5$，20～40 语句输出三个随机数。如果没有 55 语句，则 60～80 行打印结果应与 20～40 语句相同。今由于 55 语句调用 RND (-1)，使重新设置"种子"值，因此，改变了随机数序列，60～80 语句输出结果与 20～40 语句的不同。

(6) RND 函数得到的值为 (0, 1) 区间的一个数（即0.000001～0.9999999），但可以由此得到任意范围内的一个随机整数，例如：

INT (RND * 10)

可以得到一个 0～9 之间的整数。因为 RND 的值为0.0000001～0.9999999之间的一个小数，乘以 10，得到0.000001～9.999999之间的一个小数，再取整，得到一个 0～9 之间的一个随机整数。同理：

INT (RND * 100)

得到一个 0～99 之间的随机整数。

如果想得到一个 0～n 之间的随机整数，可用：

INT (RND * (n+1))

如果想得到 10～99 之间的一个随机整数，可用：

INT (RND * 90) +10

因为 INT (RND * 90) 可得到一个 0～89 之间的整数，再加 10，就得到 10～99 之间的一个随机整数。

如果想得到一个 [a, b] 间的随机整数，可用：

INT (RND * (b-a+1)) +a

【例7.6】 我们编一个"加法练习程序"，用计算机给小学生出 10 道两个二位整数的加法题目，当小学生对某一题回答正确时，计算机告诉他"GOOD!"，并加十分，再出第二题。如回答错了，计算机告诉他"YOU AER WRONG!"（你错了），并接着出下一题。回答完 10 题后计算机告诉他共得多少分。

```
10   G=0
20   FOR I=1 TO 10
```

```
30    A=INT (RND (1)  *90+10)
40    B=INT (RND (1)  *90+10)
50    PRINT A; "+"; B; "=";
60    INPUT C
70    IF C=A+B THEN PRINT "GOOD!": G=G+10 ELSE PRINT "YOU ARE WRONG!"
110   NEXT I
115   PRINT "YOUR SCORE IS"; G
120   END
```

运行记录如下：

```
62+18=? 80↙
GOOD!
15+27=? 42↙
GOOD!
23+98=? 121↙
GOOD!
39+52=? 91↙
GOOD!
18+78=? 96↙
GOOD!
```
（下接右栏）

```
76+26=? 82↙
YOU ARE WRONG!
18+78=? 86↙
YOU ARE WRONG!
87+32=? 119↙
GOOD!
43+65=? 108↙
GOOD!
57+92=? 149↙
GOOD!
YOUR SCORE IS 80（你的成绩是 80 分）
```

A 和 B 是二位数字的随机整数，如果输入（回答）的 C 值等于 A+B 的值，则 G 值加 10，否则不加分。

**【例7.7】** 发售有奖储蓄，每 10 万为一组，设一等奖一个，二等奖三个，三等奖 10 个。用计算机开奖，打印出得获者号码。

```
10   RANDOMIZE
20   A=INT (RND (1)  *10000) +1
30   PRINT "CLASS 1:"
40   PRINT A
50   PRINT "CLASS 2:"
60   FOR I=1 TO 3
70     B (I) =INT (RND (1)  *10000) +1
80   PRINT B (1),
90   NEXT I
100  PRINT : PRINT
110  PRINT "CLASS 3:"
120  FOR I=1 TO 10
130    C (I) =INT (RND (1)  *10000) +1
140    PRINT C (I),
150  NEXT I
160  PRINT
170  END
RUN
Random number seed (-32768 to 32767)? 5↙
CLASS 1:
```

```
354
CLASS 2:
9371          8689          3503
CLASS 3:
5134          8901          991          6425          9386
9587          1115          2013         7245          4643
```

每次运行程序时可输入不同的“种子”，以得到不同的随机数序列。

随机函数可用来模拟实际生活中的随机事件，例如工厂若要抽查产品质量，教师要从试题库中随机地抽取试题，都可以用计算机模拟方法来处理。

**【例7.8】** 有红、黄、蓝、白、黑 5 个球，按任意顺序一个一个取，问：按红-黄-蓝-白-黑这样的顺序出现的概率是多少？

我们以整数 1，2，3，4，5 分别代表红黄蓝白黑五种颜色。程序可编写为：

```
10  INPUT " Number="; M
20  FOR I=1 TO M
30    C1=INT(RND(1)*5)+1
40    C2=INT(RND(1)*5)+1
50    C3=INT(RND(1)*5)+1
60    C4=INT(RND(1)*5)+1
70    C5=INT(RND(1)*5)+1
80    IF C1=1 AND C2=2 AND C3=3 AND C4=4 AND C5=5 THEN N=N+1
90  NEXT I
100 PRINT N/M
110 END
RUN
Number=1000↙
 .001
```

BASIC 的函数很丰富，读者可自己根据需要选用，例如可以利用 TAB 函数打印由字符组成的图案（见第八章程序举例）。

## §7.2 自定义函数

有时在程序中需要多次用到某一公式，或要处理某一函数关系。例如，求不同半径时圆的面积，没有现成的标准函数可以使用，程序设计者可以自己定义一些所需的函数并调用它们。这就是 BASIC 的自定义函数，用 DEF 语句来定义一个自定义函数。定义函数语句的一般形式为：

**DEF FNa（参数）＝表达式**

DEF 是 Define（定义）的缩写，FN 是 Function（函数）的缩写。a 是用户自己定义的名字。此名字遵循变量名的定名规则。它和 FN 一起构成一个函数名。

**【例7.9】** 求圆的周长、圆面积、圆球体积。给定半径分别为 1，2，3，4，5。

```
10  PI=3.141593
20  DEF FNL(R)=2*P1*R
```

```
30   DEF FNS(R)=PI * R * R
40   DEF FNV(R)=4/3 * PI * R∧3
50   PRINT " R"," CIRCUM"," AREA"," VOLUME"
60   FOR R=1 TO 5
70     PRINT R, FNL(R), FNS(R),FNV(R)
80   NEXT R
90   END
RUN
```

| R | CIRCUM | AREA | VOLUME |
|---|---|---|---|
| 1 | 6.283186 | 3.141593 | 4.188791 |
| 2 | 12.56637 | 12.56637 | 33.51033 |
| 3 | 18.84956 | 28.27434 | 113.0974 |
| 4 | 25.13275 | 50.26549 | 268.0826 |
| 5 | 31.41593 | 78.53983 | 523.5989 |

当 R=1 时，打印 R,FNL(1),FNS(1),FNV(1)的值。其执行情况是：将实际参数的值 1 代替函数定义语句中的自变量 R。将 1 代替 R，可以计算出表达式的值，即 FN(1)=2 * PI * 1=2 * 3.141593×1,FNS(1)=3.141593 * 1 * 1,FNV(1)=4/3 * 3.141593 * 1^3，同样，当 R=2 时，将 2 代替定义函数语句中的 R，再求出 FNL(2),FNS(2),FNV(2)的值。

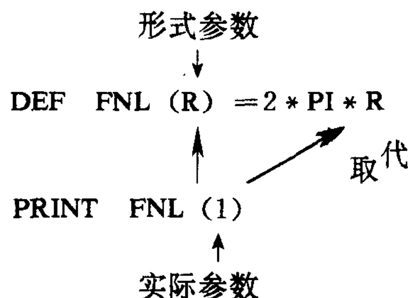
说明：(1) 定义函数只能用一个语句来完成，也就是用一个表达式来表达一个函数关系。象下面的函数关系就无法用一个表达式表示，无法在一个语句中定义。

$$y=\begin{cases}1 & (x>0)\\ 0 & (x=0)\\ -1 & (x<0)\end{cases}$$

而 $y=x^2+x+1$ 这样的函数关系是可以用一个语句定义的：

```
10  DEF FNY (x) =X * X+X+1
```

(2) 函数定义语句中的参数称为形式参数，它并不代表一个实际存在的变量，也没有固定的值。在调用此函数时它将被一确定的值所代替。如下示：




也就是说，在函数定义语句中的参数只具有形式上的作用，只有被实际参数取代后才能得到函数的值。

(3) 形式参数的名字并不重要，重要的是表达式所表示的函数关系和调用时所给定的实际参数。例如把例7.9中 20～40 语句中的 R 全改为 X，即：

```
20   DEF FNL (X) =2 * PI * X
30   DEF FNL (X) =PI * X * X
40   DEF FNV (X) =4/3 * PI * X∧3
```

程序运行结果完全相同。

(4) MS BASIC 允许形式参数有若干个，例如，求 $z=\sqrt{x^2+y^2}$，可以这样定义：

```
10  DEF FNZ (X, Y) =SQR (X*X+Y*Y)
```

(5) DEF FN 语句不是执行语句，只是定义函数关系，20 语句～40 语句不产生具体操作，只有到 60 语句调用函数时才利用该函数关系计算出函数的值。

(6) 参数和函数值可以是数值型的，也可以是字符串类型（有关字符串类型见第八章）。

(7) 如果在 DEF FN 语句中所用参数名与程序中所用变量名相同，它们并不代表同一对象，如：

```
10  DEF FNY (X) =X*X+1.2*X+2.5
20  X=3
30  PRINT FNY (2), X
```

注意：20 语句中的变量 X 不影响 10 语句中的 X，不意味着：$y(3)=3^2+1.2\times3+2.5$，30 语句显示的 FNY(2)的值只受实际参数 2 影响，即 $y(2)=2^2+1.2\times2+2.5$。

(8)如果 DEF FN 语句中表达式中使用的变量名不出现在参数表中，则它代表程序中实际存在的变量，在调用该函数时应提供该变量的值，如

```
10  DEF FNY (X) =A*X*X+B*X+C
```

反映 $y(x)=ax^2+bx+c$ 这样一个函数关系，$y$ 只是 $x$ 的函数。A，B，C 并未出现在形参表中，它们不作为形参。它们作为一般变量。如果在 10 语句后面还有以下语句：

```
20  A=2: B=3: C=4
30  PRINT FNY (5)
```

此时 FNY (5) 的值为：$2\times5^2+3*5+4=2\times25+15+4=69$

(9) 函数定义语句中可以出现另一已定义的函数，如给三角形三边求面积，可以：

```
10  DEF FNS (A, B, C) = (A+B+C)/2
20  DEF FNAREA(A,B,C)=SQR(FNS(A,B,C) * (FNS(A,B,C)-A) * (FNS(A,B,C)-B) * (FNS(A,
    B,C)-C)
30  PRINT FNAREA (3, 4, 5)
```

在 20 语句中用到已定义的函数 FNS(A，B，C)。在 30 语句引用 FNAREA 函数时，在计算 FN AREA (3，4，5) 过程中又引用函数 FNS (A，B，C)。

可以看到：引用自定义函数的方法与引用标准函数的方法几乎完全一样，只是标准函数是已由系统定义好的而自定义函数需要用户自己定义。

**【例7.10】** 将两条曲线打印在同一坐标图上。

$y1(x)=\sin x$

$y2(x)=2\cos(2x+30°)$

打印出的图形见图7.1示。y1 (x) 曲线用字符 “x” 表示，y2 (x) 曲线用 “0” 字符表示。每隔 10°打印一个点，中轴线取在 30 列上，曲线 “振幅” 定为 10 列。

处理本题的思路是：对某一个 $x$ 值，计算出 y1 (x) 和 y2 (x) 的值，分别乘 10（振幅扩大 10 倍，否则 sin（$x$）的变化范围只能从－1 到 1，变化太小无法显示出来)，再加上 30（将中心线右移到第 30 列处)。

程序如下：

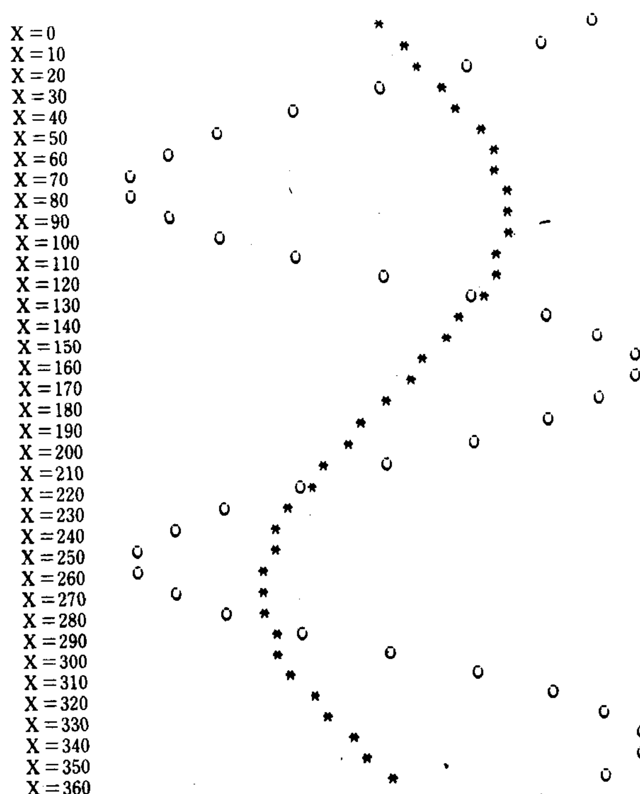



图 7.1

```
10  DEF FNY1 (X) =SIN (X)
20  DEF FNY2 (X) =2*COS (2*X+3.141593/6)
30  FOR X=0 TO 2*3.141593 STEP (3.141593/18)
40  PRINT " X="; INT (X*180/3.14159);
50  F1=INT (10*FNY1 (X) +0.5) +30: F2=INT (10*FNY2 (X) +0.5) +30
60  IF F1=F2 THEN PRINT TAB(F1);"*" ELSE IF F1>F2 THEN PRINT TAB(F2);"0";TAB(F1);"
    *" ELSE PRINT TAB(F1);"*";TAB(F2);"0"
70  NEXT X
80  END
```

程序中用自定义函数，若改为打印其它两种曲线，只需改10和20语句即可。50语句是计算应该在一行上哪一列上打印“*”或“0”字符，对计算出的函数值FNY1 (X) 和FNY2 (x) 分别乘10，四舍五入，加30。如果F1=F2，表示两条曲线在此点重合，只能打印一种字符，今令其打印“*”。如果F1>F2，表示在一行上应先打印y2 (X) 曲线（字符“0”）后打印y1 (x) 曲线（字符“*”），如果F1<F2，则应先打印y1 (x)，后打印y2 (x)。

## §7.3 子 程 序

### 7.3.1 子程序的概念

在结构化程序设计中，提倡使用“模块化”方法；把一个大的任务分解为若干个子任务，

每一个子任务就是一个模块。在程序中，用子程序来实现一个模块的功能。这样可以使程序清晰，容易编写。一个子程序可以被程序多次调用。

有时程序中的某部分语句需要多次重复执行。譬如需要多次打印一排星号“*”。其程序可写为：

```
10   FOR I=10 TO 49
20     PRINT TAB (I); "*";   } 打印 1 排（四十个）"*" 号
30   NEXT I
⋮
100  FOR I=10 TO 49
110    PRINT TAB (I); "*";   } 再打印 1 排 "*" 号
120  NEXT I
```

可以将打印一排星号的一段程序（100—120 语句）编成一个子程序。在程序需要的地方写上调用子程序语句，每调用一次就打印一排星号。如：

```
10   GOSUB 1000
20
⋮
100  GOSUB 1000
⋮
1000 FOR I=10 TO 49
1010 PRINT TAB (1); "*";     } 打印一排星号的子程序
1020   NEXT I
1030 RETURN
⋮
```

其中 1000～1030 语句就是子程序。10 语句和 100 语句（GOSUB 语句）是调用子程序语句，GOSUB 意思是“转去执行子程序”(GO SUBPROGRAM)。GOSUB 1000 表示转去执行第一个语句的行号为 1000 的子程序。子程序以 RETURN 语句结束。当程序执行到 RETURN 语句时，就返回到主程序去，接着执行调用语句的下一个语句。如果是由 10 语句调用子程序的，则执行完子程序后接着执行 10 语句的下一语句，即 20 语句。

### 7.3.2 子程序的结构和子程序的调用

在 BASIC 中子程序是与主程序连写在一起的。子程序并无明显的开始标志，即从程序语句形式来看对子程序中第一个语句没有任何特殊规定，那末子程序的起始范围是如何确定的呢？子程序的起点是由 GOSUB 语句决定的。例如上面程序，10 语句是“GOSUB 1000”，则此时调用的子程序是从 1000 语句开始，直到遇到 RETURN 语句就结束子程序的操作，假如有以下程序段：

```
10     GOSUB    1000
⋮
50     GOSUB    1100
⋮
1000    REM     SUB1
```

```
⋮
1000    REM    SUB2
⋮
1200    RETURN
```

10 语句调用的子程序是由 1000～1200 语句组成的，而 50 语句调用的子程序则是由 1100～1200 语句组成的。

调用子程序必须用 GOSUB 语句，GOSUB 语句的一般形式为：

**GOSUB 〈子程序第一个语句行的行号〉**

调用子程序时是按下面的顺序执行的：①转去子程序的第一个语句；②执行子程序中的语句；③返回主程序；④继续执行主程序；前面7.3.1段中的程序调用二次子程序（而且是同一个子程序），其执行过程如图7.2所示，图中按①～⑧顺序执行，其中①～④的含义如上述。⑤～⑧的作用如下：⑤再次调用子程序，转去子程序的第一个语句；⑥执行子程序中的语句；⑦返回主程序；⑧继续执行主程序。

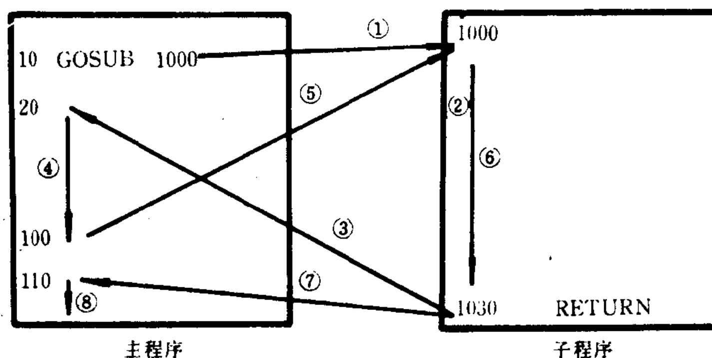



图 7.2

【例7.11】 验证哥德巴赫猜想：一个不小于 6 的偶数可以分解为两个素数（质数）之和，例如：6＝3＋3，8＝3＋5，10＝3＋7，……。输入任一偶数 $N$，要求打印出它是由哪两个素数组成。

解此问题的思路是这样的：假如给一个偶数 18，可以把它分解为两个数 N1 和 N2，N1＋N2＝N＝18。先使 N1＝2，判断 N1 是否素数，2 是素数，求 N2＝N－N1＝18－2＝16，再判断 N2 是否素数，16 不是素数，则 2 和 16 这一对数不符合要求。再使 N1 变成 3，3 是素数，N2＝N－N1＝18－3＝15，15 不是素数，这一对数也不符合要求。N1 再变成 4，4 不是素数，这时不必再求 N2 和判断 N2 是否素数。再使 N1＝5，5 是素数，N2＝N－N1－18－5＝13，13 是素数，5 和 13 都是素数，因此，18＝5＋13 是符合题目要求的。以上过程见图7.3。

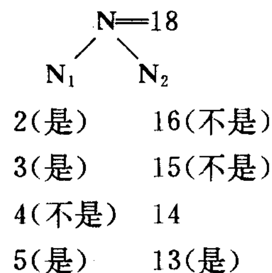



图 7.3

根据以上思路，可以设计出算法：

S1：输入 N

S2：1⇒N1

S3：使 N1＋1⇒N1，直到 N1 是素数

S4：N2=N−N1

S5：若N2不是素数，返回S3。否则打印N=N1+N2

以上算法可用图7.4表示。

但这个算法还没有完全细化，“求素数”还应进一步细化。我们可以用一个子程序来判断一个数是否素数。可以这样考虑，子程序对一个数判断其是否素数，若是素数，则使变量F=0，若不是素数则使F=1。在主程序中判断F是0还是1即可。

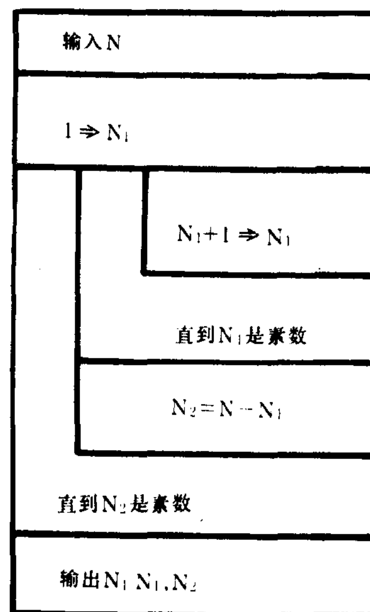



图 7.4

程序可写如下：

```
10 INPUT", N=", N
20   N1=1
30   N1=N1+1
40   M=N1
50   GOSUB 1000
60   IF F=1 GOTO 30
70   N2=N-N1
80   M=N2
90   GOSUB 1000
100   IF F=1 GOTO 30
110   PRINT N;" =";N1;" +";N2
120   GOTO 1080
1000   REM SUBPROGRAM
1010   F=0
1020   IF M=2 THEN RETURN
1030   J=INT (SQR (M))
1040   FOR I=2 TO J
1050   IF M/I=INT (M/I) THEN F=1
1060   NEXT I
1070   RETURN
1080   END
RUN
①N=18↙
  18= 5 + 13
②N=20↙
  20= 3 + 17
```

请注意：

（1）子程序是对M作判断是否素数。为什么在子程序中以M为对象而不是直接对N1或N2进行素数判断？这是为了提高子程序的通用性。第一次调用子程序时，目的是判断N1是否素数，因此在调用子程序之前，应先将N1的值赋给M。如果M是素数，则从子程序带回的变量F的值为0，否则F为1。60语句就根据F是0或1作相应处理。第二次调用子程序的目的是判断N2是否素数，因此，和语句先将N2赋给M。若执行子程序后F=1，表示M不是素数（即N2不是素数），应返回30语句，使N1的值加1，再重新进行以上步骤。

（2）请注意120语句的作用，如果无此GOTO语句，则执行完110语句打印出结果后，还会执行子程序中的语句。这一点应十分注意。要将子程序与主程序分隔开，使主程序只能

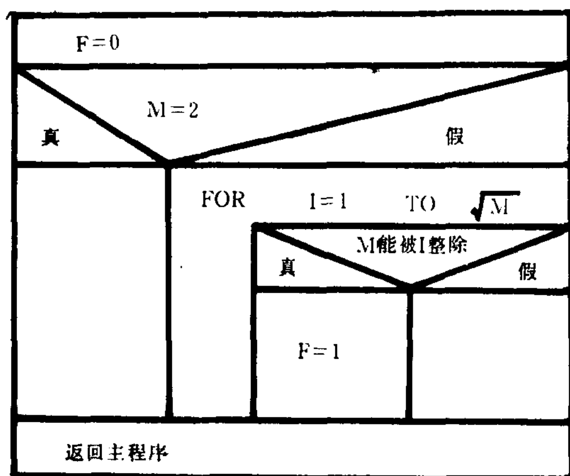



图 7.5

用 GOSUB 语句调用子程序，而避免不经过 GOSUB 而直接进入子程序（这时再遇 RETURN 就会出现错误）。因此要将子程序和主程序隔离开来（用 GOTO，STOP 或 END 语句来隔离）。

(3) 子程序中有两个 RETURN，无论遇哪一个 RETURN 语句都使得结束子程序过程而使流程返回主程序。一个子程序中可以有多个 RETURN，无论执行到哪一个 RETURN 语句都立即返回主程序。子程序的范围为 1000～1070 语句，不要误认为子程序的范围为 1000～1020 语句。子程序的操作可以用图7.5表示。1020 语句也可以改为：

1020 IF M=2 THEN GOTO 1070

即汇合到一个 RETURN 语句处返回主程序。

### 7.3.3 子程序的嵌套

在执行一个子程序的过程中还可以调用另一个程序，在执行第二个子程序的过程中，又可以调用第三个子程序。这样可以一个一个地调用下去。这种调用称作子程序的嵌套，见图 7.6。图中所示为一个主程序，以及三个子程序嵌套，其执行的顺序为：①从主程序转去执行第一个子程序；②执行子程序 1，直至遇到 GOSUB 语句；③转去第二个子程序；④执行子程序 2，直到遇到 GOSUB 语句；⑤转去第三个子程序；⑥执行第三个子程序。此子程序中不再嵌套新的子程序。因此一直执行到 RETURN 语句，⑦返回子程序 2 中调用子程序 3 的 GOSUB 语句的下一句；⑧继续执行子程序 2 的语句，直到遇 RETURN 语句；⑨返回调用子程序 2 的第一个子程序中的 GOSUB 语句的下一个语句；⑩继续执行子程序 1 的语句，直到遇 RETURN 语句；⑪返回主程序中调用子程序 1 的下一语句；⑫继续执行主程序中的语句。

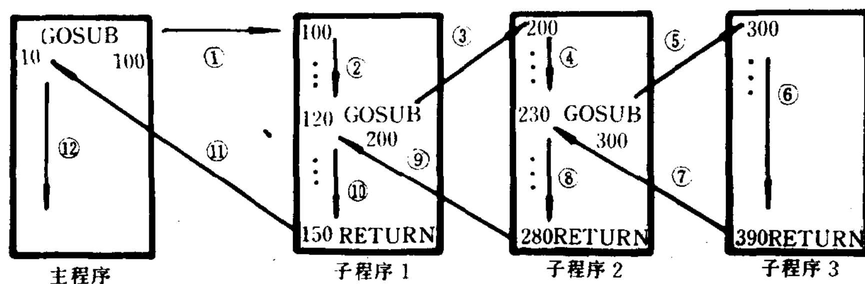



图 7.6

如果程序中用到子程序的嵌套，应注意每个子程序中的 RETURN 语句使流程返回到哪一个子程序（或主程序），请仔细观察图7.6的箭头指向。应该是：返回到调用该子程序的调用语句的下一句去。例如子程序 3 是由子程序 2 调用的，因此，执行到子程序 3 的 RETURN 语

句时，应返回到子程序 2，而不应返回到主程序或子程序 1。

与图7.6相对应的程序可写为：

```
5     PRINT "0"          (尚未调用子程序)
10    GOSUB 100          (调用子程序 1)
30    END                (主程序结束)
100   PRINT "1"          (子程序 1 开始)
110   GOSUB 200          (调用子程序 2)
120   RETURN             (返回主程序)
200   PRINT "2"          (子程序 2 开始)
210   GOSUB 300          (调用子程序 3)
220   RETURN             (返回子程序 1)
300   PRINT "3"          (子程序 3 开始)
310   RETURN             (返回子程序 2)
```

执行结果为：

```
RUN
0          (执行主程序时打印出来的)
1          (执行子程序 1 时打印的，表示进入第 1 层子程序)
2          (执行子程序 2 时打印的，表示进入第 2 层子程序)
3          (执行子程序 3 时打印的，表示进入第 3 层子程序)
```

可以看到，子程序还可以调用另一子程序，一个程序中只能有一个主程序，却可以有多个子程序。子程序既可以被调用，也可以调用其它子程序。不要认为凡是调用子程序的都叫主程序。确切地说，应当称为“调用程序”和“被调用程序”。主程序只能是“调用程序”，而子程序既可以作为“调用程序”，也可以作为“被调用程序”。

子程序的嵌套最多可以多少层（称为嵌套深度），不同的 BASIC 有不同的规定，MS BASIC 不限制嵌套层数，只受到内存空间的限制（嵌套层次多占内存愈多），有的 BASIC 只允许嵌套 8 层。请注意，下面所示的不是嵌套。

```
10    GOSUB 100
20    GOSUB 100
30    GOSUB 100
```

它只是连续多次调同一个子程序，在调用一次子程序结束之后才开始另一次调用子程序的操作。

子程序直接或间接调用自己，称为递归调用。如：

```
10 GOSUB 10
```

是不行的。也不要象以下这样使用：

```
10 GOSUB 100
  ⋮
100 GOSUB 10
```

也是不行的，因为在子程序中又反过来调用子程序，永远不会结束。见图7.7。

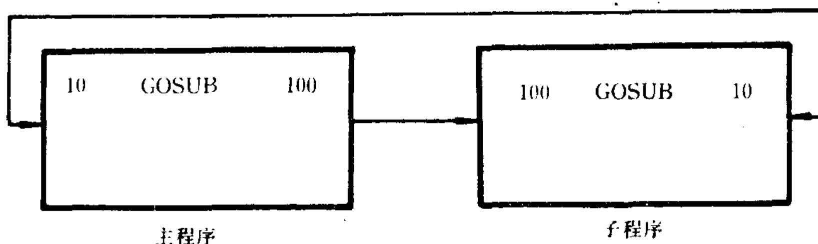



图 7.7

但是如果在递归调用时能使调用在一定限次数内结束，则是允许的。例如：

```
10 X=3
20 PRINT X;
30 GOSUB 100
40 END
100 X=X-1
110 PRINT X;
120 IF X=0 THEN
    RETURN
130 GOSUB 100
140 RETURN
```

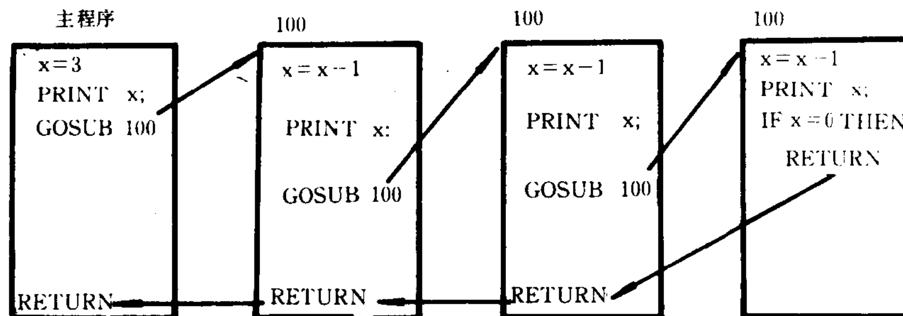



图 7.8

这是子程序间接调用自己。在执行 100～140 子程序过程中又调用子程序 100～140。其调用过程可以用图7.8示。

图7.8中，在前二次调用中由于 $X>0$ 所以未画出“IF X=0 THEN RETURN”的操作，以便简明清晰。在第一次调用时，$X=3-1=2$，打印出“2”，第二次调用本子程序，$X=2-1=1$，打印出“1”，第三次调用本子程序，此时 $X=1-1=0$，打印出“0”，然后返回，请注意每次返回的返回点。

```
RUN
 3  2  1  0
```

可以用跟踪命令 TRON 打印出执行语句的情况：

```
TRON↙
RUN↙
[10] [20] 3 [30] [100] [110] 2 [120] [130] [100] [110] 1 [120] [130] [100]
[110] 0 [120] [140] [140] [40]
```

在 BASIC 中使用递归调用虽然是允许的，但不容易搞清其调用关系，故不提倡初学者使用递归调用。

### 7.3.4 IF-GOSUB 语句和 ON-GOSUB 语句

我们已经介绍过 IF-GOTO 语句和 ON-GOTO 语句，BASIC 还提供 IF-GOSUB 语句和 ON-GOSUB 语句，其语句一般形式为：

**IF<逻辑表达式>GOSUB<行号>**

**ON<算术表达式>GOSUB<行号 1，行号 2，…>**

例如：

```
10 IF X>0 GOSUB 100
20 ON X GOSUB 200, 300, 400, 500
```

它们与 IF-GOTO 语句和 ON-GOTO 语句的区别在于：执行完 GOSUB 语句后返回到 IF-GOSUB 语句或 ON-GOSUB 语句的下一个语句，利用 ON-GOSUB 语句可以在应用程序中设计屏幕菜单供用户方便使用（见第十章）

## 7.3.5 子程序的作用

子程序的作用是专门用来完成某一指定的功能的。例如：求阶乘、找素数、排序、打印一组信息……等。可以根据不同的需要编写出不同的子程序。当然也可以不用子程序，而直接在主程序中适当的地方写出这个程序段。但是这就使主程序段显得冗长，看起来费劲难懂。

有人将常用的一些需要求解的问题（例如解联立方程，求定积分，排序……）写成一个个子程序，汇编成册，用户需要时，可直接抄用代入程序即可，不必每次自己再编写这一段程序了。也可以把这些子程序分别指定名字（即文件名）存在磁盘上，需用时按文件名调到内存，然后用 GOSUB 调用，这就是“程序库”。在解决复杂问题时，事先编好的子程序是十分有用的。

如果一个程序段多次被调用，显然把它写成子程序是方便的，可以提高编程序的效率。

有时，尽管某一程序段只需要调用一次，也把它写成子程序，并把程序应该完成的主要功能都分配给各子程序去完成。这样主程序可以写得比较短，用一个 GOSUB 语句调用一个子程序。主程序就主要由若干个 GOSUB 语句组成了。如：

```
10    GOSUB 100
20    GOSUB 200
30    GOSUB 300
40    END
```

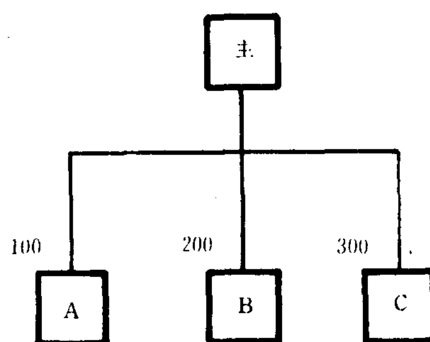


图 7.9

可把主程序和各子程序看作一个一个的模块，主程序可以看作控制模块，子程序可以看作功能模块。主程序好比总调度，调各功能模块完成各部分功能。见图7.9。这种模块化程序设计方法可以使程序逻辑清楚，易于阅读和理解，也便于调试。

## 习　　题

利用 BASIC 提供的标准函数完成习题7.1～7.9。

7.1 编写程序，打印出 0～100 之间能被 3 和 5 同时整除的整数。

7.2 编写一个“一位数乘法练习程序”，打印出题目：A * B=？(A 和 B 各为一个 0～9 之间的整数)。根据回答的结果，给出“答对了”或“答错了”的信息。若答对则再出另一题目，若答错，重新打印出该题目，要求重做，直到做对为止。出题目是反复地进行的。

7.3 以下程序会产生什么打印结果：

```
10   FOR X=1 TO 8
20   PRINT TAB (10-X);
30   FOR A=1 TO 2 * X-1
40   PRINT "*";
50   NEXT A
60   PRINT
70   NEXT X
99   END
```

7.4 想打印如图 7.10 所示的菱形图案，请编程序。

7.5 输入十个整数，要求分别按奇数和偶数打印出来，并分别求出奇数和以及偶数和。

7.6 求最初的 100 个素数。

7.7 模拟玩骰子游戏，每个骰子为一个正六面体，每一面分别刻有 1,2,3,4,5,6 个点。假设有 4 个骰子，问同时出现 4 个“6 点”的机率是多少？

7.8 有三堆球，每堆球中都有红、黄、蓝、白、黑五种颜色的是五个球，今随手从每堆中各取一个球，如果共取 1000 次，问：①取到三种相同色的球的次数是多少？②有两个球颜色相同的次数；③三个球都不同色的次数。

```
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
 * * * * * * *
  * * * * *
   * * *
    *
```

图 7.10

7.9 用自定义函数编程序，求三角形面积。三角形三边 $a,b,c$ 由键盘输入。

7.10用自定义函数求 $A,B$ 二点间的距离 $S$，分别求出 $A$，$B$ 为下列各坐标值时的距离。

①A (3, 5), B (5, 3)　　②A (−3, 8), B (5, 7)

③A (0, −5), B (7, 10)　　④A (−7, −8), B (−5, −6)

7.11 求多项式 $ax^2+bx+c$ 的值，a，b，c 的值分别为以下几组：①a=1，b=2，c=3；②a=1. 5，b=2. 5，c=3. 5；③a=0，b=4 c=6；④a=−5，b=−3，c=7。定义函数 f (a，b，c)，即 f 为 a，b，c 的函数。x 由键盘输入。编程序求出并打印以上 4 种情况下多项式的值（在这 4 种情况下 x 值是相同的）。

7.12 求 100～200 之间的全部素数，并打印出来，用子程序编程序。

7.13 写出下面程序运行的结果，并写出语句执行的顺序。

```
10  GOSUB 100
20  GOSUB 200
30  GOSUB 300
40  END
100  READ A, B
110  C=A+B
120  PRINT C
130  RETURN
200  READ A, B
210  C=A*B
220  PRINT C
230  RETURN
300  READ A, B
310  C=A/B
320  PRINT C
330  RETURN
900  DATA 15, 20, 25, 30, 35, 40
```

7.14 采用子程序的方法求 7! +8! +9! +10!

# 第八章 字 符 处 理

## §8.1 概　　述

前几章介绍了计算机在数值计算中的应用，前面所介绍的程序中用到的变量和数组都是数值型的。事实上，计算机的另一个十分重要的用途是非数值运算，尤其是用于事务管理领域，例如图书检索、人事管理、教务管理、经济管理等。在这些领域中需要用计算机处理一些文字（例如把以字母“A”开头的书目打印出来，按英文字母顺序排列各国的国名等）。这就要求计算机有文字处理的能力。有人说，如果一个计算机没有文字处理功能，充其量它只是一个高级计算器。

BASIC 的一个重要功能就是可以进行字符处理，而且它的字符处理能力是比较强的，它提供了字符串变量、字符串数组，以及丰富的字符串函数，使用十分灵活方便。

所谓“字符串”是由系统允许使用的若干个字符组成的。在第三章§3.2中已介绍过可以直接用 PRINT 语句输出一个“字符串”，例如：

```
100 PRINT “CHINA”，“BEIJING”
```

“CHINA”和“BEIJING”就是字符串。除了可以直接打印输出字符串之外，还可以将字符串存放在变量或数组中，这就是字符串变量和字符串数组。

## §8.2 字符串变量

存放数值的变量称为数值变量，迄今为止所用到的都是数值变量。数值变量的值是一个数值，例如：

```
10   A=3.6
```

A 就是一个数值变量，同样可以将一个字符串存放到一个变量中，例如：

```
10   A$ = “CHINA”
20   B$ = “BEIJING”
```

我们已经用赋值语句将字符串“CHINA”和“BEIJING”分别赋给了字符串变量 A$ 和 B$。在需要使用这些字符串时可直接调用字符串变量 A$ 和 B$，就如同调用数值变量一样，例如：

```
30   PRINT A$，B$
```

输出的结果如下：

```
CHINA          BEIJING
```

字符串变量名取名规则与数值变量相同，只是在最后加一个“$”，表示为字符串类型。如 A1，B1 为数值变量，而 A1$，B1 为字符串变量。

注意：

(1) 在赋值语句中，字符串要用引号括起来，如果写成下面这样是不对的：

```
10 A$ =CHINA
20 B$ =BEIJING
```

系统将把 CHINA 和 BEIJING 当作变量名而不作为字符串来处理。

(2) 在输出字符串变量的值时是不包括引号的。

(3) 应该将字符串和数值、字符串变量和数值型变量严格区别开来。例如 123 是一个数值，而"123"是一个字符串，它并不代表一百二十三，而仅代表三个字符"1"，"2"和"3"其中每个字符都是与其它字符无关的独立字符。因此字符串不能用来作算术运算。例如想用字符串"123"加字符串"456"来得到"579"是不可能的。

## §8.3 字符串的输入

除了可以用赋值语句给字符串变量赋值之外，还可以用 READ 语句和 INPUT 语句将字符串输入给字符串变量。

### 8.3.1 用 READ/DATA 语句向字符串变量赋值

用 READ 和 DATA 语句除了可以用来给数值型变量（普通变量）赋值外，也可以用来读字符串并将它输送给字符串变量。

**【例8.1】** 打印学生成绩的程序：

```
10  PRINT "NAME", " NO. ", "GRADE"
20  FOR I=1 TO 6
30  READ A$, B, C$
40  PRINT A$, B, C$
50  NEXT I
60  DATA "ZHANG", 101, "A", "WANG", 102, "B"
70  DATA "LI", 103, "A", "LING", 104, "D"
80  DATA" TAN", 105, "C", "FUN", 106, "A"
90  END
```

运行结果为：

```
NAME        NO.         GRADE
ZHANG       101         A
WANG        102         B
LI          103         A
LING        104         D
TAN         105         C
FUN         106         A
```

程序中 30 语句要求从 DATA 语句中读数据。由于 A$ 和 C$ 是字符串变量，因此在 DATA 语句中的第一和第三个数据应该是字符串而不是数值。许多 BASIC 要求在语句中将字符串用引号括起来。B 是普通数值型变量，因此 DATA 语句中第二个数据应该是数值而不能是字

符串。第一次执行 30 语句时，将字符串“ZHANG”输送给 A$，把数值 101 输送给 B，把字符串“A”输送给 C$，然后 40 语句打印出：

ZHANG　　101　　　A

如此循环六次，每一次读入一组姓名、学号、分数等级。然后打印出来。

说明：

(1) 可以在一个 READ 语句中出现数值变量和字符串变量，同样，在 DATA 语句中可以同时出现数值数据和字符串数据。

(2) DATA 语句中的数据类型应与 READ 语句中相应的变量的类型一致。

如果 DATA 语句写成下面这样是错误的：

60 DATA “ZHANG”，“101”，“A”，“WANG”，“102”，“B”

因为 30 语句中的 B 是数值变量，而读入的数据“101”和“102”是字符型的，产生“类型不一致”的错误。

(3) 有些 BASIC(包括 MS BASIC) 允许在 DATA 语句中的字符串数据不必用引号括起来。如上面例8.1程序中的 60 语句可以写成：

60　DATA ZHANG，101，A，WANG，102，B

但是，如果一个字符串中包括逗号或前导空格或尾随空格，则应该用引号把这个字符串括起来。例如：

10　READ DATE1$，DATE2$

100　DATA “MAY 1，1990”，“SEP 1，2000”

DATE1$ 的值为“MAY 1，1990”，DATE2$ 的值为“SEP 1，2000”，而如果在 DATA 语句中不用引号，写成：

100　DATA MAY 1，1990，SEP 1，2000

则将逗号作为分隔两个数据的标志，于是 DATE1$ 的值为“MAY 1”，而 DATE2$ 的值是“1990”。又如：

10　READ MARK$

20　PRINT MARK$

100　DATA “␣␣␣ * * * ␣TABLE␣ * * * ␣␣␣”

20 语句打印出：

␣␣␣ * * * ␣TABLE␣ * * * ␣␣␣

如果 100 语句中不用引号，写成：

100 DATA ␣␣␣ * * * ␣TABLE ␣ * * * ␣␣␣

由于在 DATA 语句中一个数据的前导空格和尾随空格是不起作用的，因此，20 语句打印结果为：

* * * ␣TABLE ␣ * * *

前导空格和尾随空格都没有了。为安全起见，建议初学者对所有字符串都加上引号。

### 8.3.2　用 INPUT 语句给字符串赋值

INPUT 语句可以用来输入数值数据或字符串数据。用法与前基本相同。

【例8.2】登记姓名和地址

```
10   INPUT "WHAT IS YOUR NAME"; A$
20   INPUT "WHAT IS YOUR STREET ADDRESS"; B$
30   PRINT
40   PRINT "NAME:", A$
50   PRINT "ADDRESS:", B$
60   END
RUN
WHAT IS YOUR NAME? "WANG HAO" ↙
WHAT IS YOUR STREET ADDRESS? "192 BEIJING ROAD" ↙
NAME: WANG HAO                }
ADDRESS: 192 BEIJING ROAD     } 打印结果
```

从键盘回答询问，将字符串敲入时，一般应该用引号将字符串括起来。如上面所示。但是如果字符串中不包括逗号、前置空格或尾随空格时，引号亦可以省去。如上面程序运行时，也可以输入如下：

```
RUN
WHAT IS YOUR NAME? WANG HAO↙
WHAT IS YOUR STREET ADDRESS? 192 BEIJING ROAD↙
```

如果有：

```
10 INPUT "DATE"; D$
```

执行时，以下回答不合适：

```
RUN
DATE? MAY 1, 1990↙
```

D$的值是“MAY 1”而不是“MAY 1，1990”。我们建议，初学时最好把每个字符串都用引号括起来，以免引起错误。

可以在一个INPUT语句中既使用字符串变量，又使用数值变量。在键盘输入数据时，应相应地分别输入字符串和数值。如：

```
10   INPUT "ENTER YOUR NAME AND AGE"; A$, B
20   PRINT B, A$
99   END
RUN
ENTER YOUR NAME AND AGE? "ZHANG FUNG", 20↙
20              ZHANG FUNG
```

## §8.4 字符串表达式

可以把两个或多个字符串或字符串变量连接起来。BASIC中用“+”作为连接字符串的运算符（称为“字符串运算符”）。

"THIS" + "␣IS" + "␣A" + "␣BOOK."

字符串运算符是对字符串数据进行运算的唯一的运算符，它的作用是把“+”运算符两侧的

字符串连接成一个字符串，用“＋”号将字符串数据连接起来的式子称“字符串表达式”。上面就是一个字符串表达式。“＋”两侧可以是字符串、字符串变量、字符串数组元素、或值为字符串的串函数（有关串数组和串函数将在本章稍后介绍）。

可以将一个字符串表达式的值赋给一个字符串变量（或字符串数组元素）。如：

```
10 A$ = "THIS" + "␣IS" + "␣A" + "␣BOOK."
```

A$的值为："THIS IS A BOOK."

## §8.5 字符串的比较

两个数值可以比较，如5>3，X>Y等。两个字符串（或字符串变量）也可以比较。

如果两个字符串，它们的每一个字符（包括头尾的空格）都相同，则认为这二个字符串相等。

**【例8.3】**

```
20  INPUT "ARE YOU A BOY"; A$
30  IF A$ = "YES" PRINT "YOU ARE A BOY" ELSE PRINT "YOU ARE A GIRL"
40  END
RUN
ARE YOU A BOY? "YES" ↙
YOU ARE A BOY
```

由于在执行20语句时已将“YES”输入给A$，因此IF语句中关系表达式的值为真，打印出“YOU ARE A BOY”。如果输入给A$的值不是“YES”，而是“NO”，则在执行IF语句时判断出A$不等于“YES”，执行ELSE后面的语句，打印出“YOU ARE A GIRL”。

注意，字符串中的空格也作为一个字符参加比较：例如“YES”不等于“Y ES”，因为后者Y和E之间有一空格。

**【例8.4】** 可以编一个寻找图书的程序。在DATA语句中先放入若干本书的名字、编号和作者，如果想查询有没有某一本书，可以从键盘输入此书名字，如有此书，则会打印出此书的名字和书号及作者。如果找不到此书，则打印出“CAN’T FIND THIS BOOK!”。

程序如下：

```
10   INPUT "TYPE NAME, THEN HIT ENTER"; A$
20   RESTORE
30   FOR I=1 TO 6
40     READ N, B$, C$
50     IF B$ =A$ THEN I=10
60   NEXT I
70   IF I>=10 THEN PRINT N, B$, C$ ELSE PRINT "CAN'T FIND THIS BOOK!"
80   INPUT "TO CONTINUE TYPE YES; ELSE NO."; D$
90   IF D$ = "YES" THEN GOTO 10
100    DATA 10519, "ENGLISH", "ZHANG"
110    DATA 20418, "PHYSICS", "LI"
120    DATA 30123, "CHEMISTRY", "FUN"
```

```
130    DATA 10253, "MATHEMATICS", "WANG"
140    DATA 46352, "CHINESE", "TAN"
150    DATA 52312, "PHILOSOPHY", "SAN"
160    END
RUN
TYPE NAME, THEN H1T ENTER? "CHEMISTRY" ↙
30123              CHEMISTRY    FUN
TO CONT1NUE TYPE YES; ELSE NO. ? "YES" ↙
TYPE NAME, THEN HIT ENTER? "COMPUTER" ↙
CAN' T FIND THIS BOOK!
TO CONTINUE TYPE YES; ELSE NO. ? "NO" ↙
(结束)
```

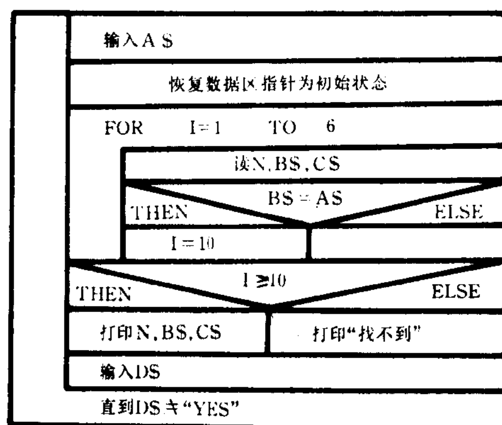



图 8.1

流程图见图8.1。

执行的过程是：从键盘输入一书名，30—60 语句是一个循环，它每次从 DATA 语句中读入一个书号给 N，一个书名给 B$，一个作者名给 C$。每一次将 A$（键盘输入的书名）和 B$（DATA 语句中事先存贮的书名）相比，如果第一次循环时 A$≠B$，则读入第二组数据（110 语句），再看 A$ 与 B$ 是否相等。如果相等，则使 I=10，这标志着“已找到”，然后使循环终止。如果在循环六次中 A$ 都不等于 B$，也终止循环，但此时 I<10。70 语句根据 I 是否小于 10 决定显示内容。如果 I>=10，则显示出书号、书名和作者名。否则显示出“找不到”。今为节省篇幅起见，只存入六本书，当然可以存 100 本或更多的书备查，但 50 语句中 I 的值也应作相应修改。

无论找到或找不出此书，在查完一本书之后，执行 100 语句，显示出：

TO CONTINUE TYPE YES; ELSE NO. ?

如果打入“YES”，表示还要查询其它的书，90 语句检查 D$ 是否等于“YES”，若相等，返回执行 10 语句，开始另一本书的查询，并使数据区指针恢复到初始位置（即第一个 DATA 语句中第一个数据之前，即 10519 之前）。如果不打入“YES”，而打入“NO”或其它字符串，则表示不再查询了。程序结束。

下面介绍一下字符串比较的规则。

可以有以下几种字符串“比较”的方式：

(1) 字符串与字符串的比较。如“YES” <> “NO”

(2) 字符串与字符串变量的比较，如 D$ = “YES”

(3) 字符串变量与字符串变量的比较，如 A$ = B$

我们在上面的程序中用到了“等于”（=）比较符。这是最简单的情况。可以用于字符串比较符有：

= (等于)

<> (不等于)

> (大于)

< (小于)

>= (大于或等于)

<=（小于或等于）

两个字符串相比较，怎么判断谁大谁小呢？在 BASIC 中，字符是用 ASCII 代码形式存入计算机中的。字符和 ASCII 码的对应关系见附录Ⅰ。字符“A”的 ASCII 代码是十进制的 65（二进制为 01000001），“B”的代码为 66（二进制码为 01000010）。字符的比较是比较它们的 ASCII 代码，以 ASCII 码大者为大。由于“A”的代码 65 小于“B”的代码 66，因此，“A”<“B”。同理，字符“1”的 ASCII 代码为十进数 49（二进数 00110001），因此“1”<“A”。

字符大小的次序可以用下表简单地表示（在每一行中左面的小，右面的大）：

空格 ! ” # $ % & ’ ( ) * + , — . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z ^

两个字符串相比，是将两个字符串的字符从左到右逐个相比，直到两个字符串中有一个不相同的字符为止，以这时的字符比较结果为准。如：

“THAT”和“THE”，它们的第一、二个字符相同，而第三个字符比较的结果是“E”>“A”，则认为“THE”>“THAT”，而不管第四个字符或其以后字符比较的结果如何。因此，“LAGLE”<“LANGE”。

如果两个字符串相应的字符都相同，而其中一字符串长一些，则以长者为大。如

“WHOSE”>“WHO”，　　　“YOUR”>“YOU”。

比较简单的记法是：按英文字在字典中的位置，在后面的值为大。如在字典中，“THESE”在“THAT”之后，因此，“THESE”>“THAT”。数字字符的比较规则与习惯相同。即“8”>“6”，“1”>“0”。

**【例 8.5】** 有若干个国家的名字，找出按英文字母排列在前面的国名。

```
10  READ L$
20  FOR I=1 TO 7
30      READ A$
35      IF A$<L$ THEN L$=A$
40  NEXT I
50  PRINT L$
70  DATA "CHINA","JAPAN",CHNANDA","KOREA"
80  DATA "ENGLAND","FRANCE","AMERICA","INDIA"
90  END
RUN
AMERICA
```

今假设只有 8 个国名，先读入第一个国名赋给 L$，然后在每一次循环中读入一个国名给 A$，若读入的 A$<L$，则以 A$ 的值代替 L$ 的原值，L$ 中始终是已读入的国名中“最小”的，在执行各次循环时 L$ 和 A$ 的值变化如表 8.1 所示：

表 8.1

| 循环次数 | 执行 40 句时 | | 执行 60 句时 | |
|---|---|---|---|---|
| | L $ | A $ | L $ | A $ |
| 1 | CHINA | JAPAN | CHINA | JAPAN |
| 2 | CHINA | CANADA | CANADA | CANADA |
| 3 | CANADA | KOREA | CANADA | KOREA |
| 4 | CANADA | ENGLAND | CANADA | ENGLAND |
| 5 | CANADA | FRANCE | CANADA | FRANCE |
| 6 | CANADA | AMERICA | AMERICA | AMERICA |
| 7 | AMERICA | INDIA | AMERICA | INDIA |

## § 8.6 字符串数组

许多 BASIC（包括 M BASIC，Applesoft BASIC，TRS－80 BASIC）允许使用字符串数组。其概念与数值数组相仿，只是每个数组元素中存放的不是数值，而是一个字符串。对字符串数组同样用 DIM 语句定义数组的维数和元素个数。

**【例 8.6】** 有一批国家名，要求按其英文字母顺序排列（以 A 为最小，Z 最大）。

今用字符串数组 A $ 来存贮国家名。

程序可编写为：

```
5   DIM A $ (10)
10   FOR I=1 TO 10
20     READ A $ (I)
30   NEXT I
40   FOR I=1 TO 9
45     L=I
50     FOR J=I+1 TO 10
60         IF A $ (J)<A $ (L) THEN L=J
70     NEXT J
80     IF L<>I THEN T $ =A $ (I):A $ (L)=T $ (L):A $ (L)T $
90   NEXT I
120   FOR I=1 TO 10
125     PRINT A $ (I),
130   NEXT I
180   DATA "CHINA","JAPAN","PHILIPPINES"
190   DATA "INDONESIA","SINGAPORE","POLAND"
200   DATA "FRANCE","ITALY","BELGIUM","MONACO"
220   END
```

请把此程序与例 6.8 数值排序的程序相比较。

5 语句是数组说明语句，今用来说明 A $ 数组的大小（一维的，有 11 个元素）。用法和数

值型数组相同。

40 至 90 语句是排序部分，它和数值排序的原理相同（读者可自己比较一下）。只是用字符串的数组元素代替了数值型数组元素。

运行记录如下：

```
RUN
BELGIUM     CHINA     FRANCE         INDONESIA     ITALY
JAPAN       MONACO    PHILIPPINES    POLAND        SINGPORE
```

**【例 8.7】** 打印图案

许多人希望利用计算机打印出由字符组成的各种图案，这种图案的程序实现起来并不难。首先用一张方格纸或坐标纸，将需要打印的图案用毛笔画好。然后从最上面一行开始，将图案上每一行中黑色的部分起止的格数记下来。在程序中要用到这些数据。

我们举一个打印熊猫图案的例子。图 8.2 所示的图案共有 77 行，第一行的黑色部分为从第 31 格到第 38 格，第二行的黑色部分为 30～39 格，……第六第的黑色部分有二处：27～30 格和 39～41 格。这些数据都依次记下来（见下面程序中 DATA 语句部分）。

我们的思路是这样的：设一个字符串二维数组 P$，它的每一个元素可以放一个字符。与图黑色部分相应位置的数组元素的值为字符“B”，其它元素置空格。最后打印出该数组即可。

从图 8.2 可以看到每一行的黑色部分都未超过 60 格，因此数组 P$ 的大小可以定为 77×60。还有一个问题要解决：如何将数据分行处理，例如第一行的数据为 31 和 38，第二行的数据为 30 和 39。我们这样处理，在每一行数据之后加一个“－1”，表示本行结束。流程图见图 8.3。

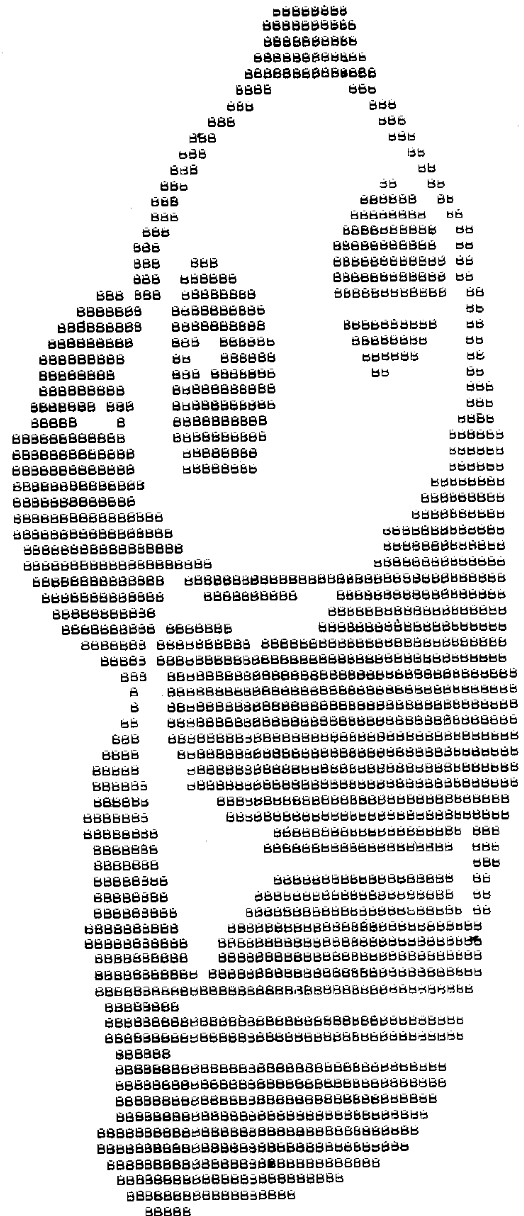


图 8.2

根据流程图可编写出以下程序：

```
10   DIM P$ (77, 60)
20   FOR I=1 TO 77
25     FOR J=1 TO 60: P$(I,J)="  ": NEXT J
30     READ N
40     IF N=-1 GOTO 110
50     READ M
60     FOR J=N TO M
70       P$ (I, J) ="B"
```

```
80     NEXT J
90     READ N
100    GOTO 40
110  NEXT I
120  FOR I=1 TO 77
130    FOR J=1 TO 60
140      PRINT P$ (I, J);
150    NEXT J
160    PRINT
170    NEXT I
185  DATA 31,38,-1, 30,39,-1,30,39,-1,29,
     40,-1,28,41,-1
190  DATA 27,30,39,41,-1,26,28,41,43,-1,24,
     26,42,44,-1
185  DATA 22,24,43,45,-1,21,23,45,46,-1
200  DATA 20,22,46,47,-1,19,21,42,43,47,48,-1,18,20,40,45,48,49,-1
205  DATA 18,20,39,46,49,50,-1,17,19,38,47,50,51,-1
210  DATA16,18,37,47,50,51,-1,16,18,22,24,37,48,50,51,-1,16,18,21,26
215  DATA 37,48,50,51,-1,12,14,16,18,21,28,37,48,51,52,-1,10,16,20,29
220  DATA 37,47,51,52,-1,8,16,20,29,38,47,51,52,-1,7,15,20,22,25,30,39
225  DATA 46,51,52,-1,6,14,20,21,25,30,40,45,51,52,-1,6,13,20,22,24,30
230  DATA 41,42,51,52,-1,6,14,20,30,51,53,-1,5,11,13,15,20,30,51,53,-1
235  DATA 5,9,14,14,20,29,50,53,-1,3,14,20,29,49,54,-1
240  DATA 3,15,21,28,49,54,-1,3,15,21,28,48,54,-1
245  DATA 3,16,47,54,-1,3,15,46,54,-1,3,18,45,54,-1,3,19,42,52,-1
250  DATA 4,20,42,54,-1,4,23,41,54,-1,5,18,21,54,-1,6,18,23,32,37,54,-1
255  DATA 7,17,36,54,-1,8,17,19,25,35,54,-1,10,16,18,27,29,54,-1
260  DATA 12,16,18,54,-1,14,16,19,55,-1,15,15,19,55,-1,15,15,19,55,-1
265  DATA 14,15,19,55,-1,13,15,19,55,-1,12,15,20,55,-1,11,15,21,55,-1
270  DATA 11,16,21,55,-1,11,16,24,54,-1,10,16,25,54,-1
275  DATA 10,17,30,49,51,53,-1,11,17,29,48,51,53,-1,11,17,32,47,51,53,-1
280  DATA 11,18,30,48,51,52,-1,11,18,28,48,51,52,-1,11,19,27,49,51,52,-1
285  DATA 10,19,25,49,50,51,-1,10,20,24,51,-1
290  DATA 11,20,24,51,-1,11,21,23,51,-1,11,50,-1,12,49-1,12,49,-1
295  DATA 12,49,-1,13,48,-1,13,47,-1,13,47,-1,13,46,-1,13,45,-1
300  DATA 11,44,-1,11,43,-1,12,40,-1,13,36,-1,14,31,-1,16,20,-1
999  END
```

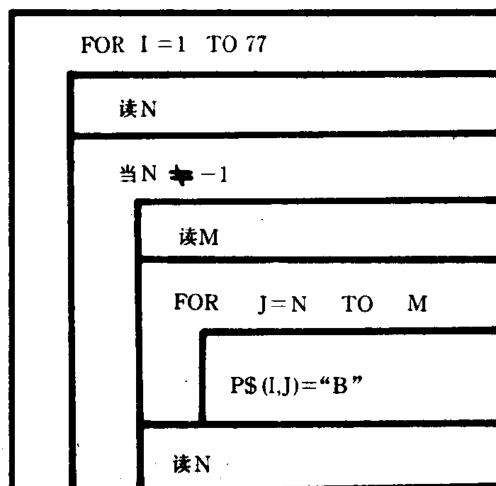



图 8.3

实际上，这个问题也可以用一维数组来处理，处理一行打印一行。请读者自己完成它。

## §8.7 子字符串

子字符串是一个字符串的一部分。例如A$的值为"ABCDE"，则"BCD"就是A$的一

个子字符串。当然一个字符串可以有多个子字符串，如“AB”，“B”，“CDE”，“DE”，“ABCD”……等都是 A $ 的子字符串。

在使用子字符串时要用到有关的“子串函数”：

**（1）LEFT $ 函数**

它的形式为：

**LEFT $ （串，n）**

它的函数值是所指定的字符串中左面 n 个字符。括弧中的“串”，可以是一个字符串，也可以是一个字符串变量或字符串数组元素，也可以是一个字符串表达式，n 可以是一个常数或算术表达式。

**【例 8.8】**

```
10  A $ = "CHINESE PEOPLE"
20  PRINT LEFT $ (A $ , 7)
30  END
```

20 语句中 LEFT $ （A $ ，7）的值是 A $ 中的左面七个字符，即 CHINESE。运行情况为：

```
RUN
CHINESE
```

**【例 8.9】** 有一批英文字，希打印出其中以 A 开头的字。

```
10  READ A $
20  WHILE A $ <> "END"
30    IF LEFT $ (A $ , 1) = "A" THEN PRINT A $
40  WEND
50  DATA "PEN", "MEN", "BOOK", "AIR", "CITY", "FEET", "COOK", "TEA", "PLANE",
   "BUS", "ARMY", "END"
99 END
RUN
AIR
ARMY
```

30 语句的作用是检查读入的字符串的第一字符是不是“A”。

**（2）RIGHT $ 函数**

它的形式为：

**RIGHT $ （串，n）**

用它求字符串中最右边的 n 个字符。如果字符串长度不超过 n，则它得到的是整个字符串。

**【例 8.10】** 某单位有十名工作人员，他们的工作证号码最后一位是表示性别，如 M 表示男性，F 表示女性，要求输入工作证号码，并统计出男、女人数。

```
5   M=0: F=0
10  FOR J=1 TO 10
15    INPUT C $
20    IF RIGHT $ (C $ , 1) = "M" THEN M=M+1 ELSE F=F+1
30  NEXT I
```

```
40  PRINT "M:"; M
50  PRINT "F="; F
60  END
```

**(3) MID$函数**

其形式为：

**MID$ (串，p，n)**

用它求一个字符串中从第p个字符开始的n个字符。如MID$ (T$，3，5) 表示取T$中从第三个字符开始的5个字符。如不指定n，则求出从第p个字符开始的后面的全部字符。

**【例8.11】** 将26个英文字母按逆序打印出来。

```
10  A$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20  FOR I=26 TO 1 STEP-1
30    PRINT MID$ (A$, I, 1);
40  NEXT I
50  END
```

第一次循环，I=26，MID$ (A$，26，1) 的值为字符"Z"，打印出"Z"，第二次循环，打印出MID$ (A$，25，1) 的值"Y"，如此一直打印出从Z到A的26个字母。

子字符串可以出现在字符串表达式中作为运算对象，即参加连接运算。

**【例8.12】**

```
10  A$ = "THIS IS A BOOK"
20  B$ = "THAT IS A PEN"
30  C$ =LEFT$ (A$, 8) +RIGHT$ (B$, 5)
40  PRINT C$
50  END
RUN
THIS IS A PEN
```

## §8.8 有关字符串运算的函数

**(1) 测LEN函数**

用LEN（串）可以求出字符串中的字符个数，即字符串的长度。

**【例8.13】** 将输入的十个字符串中长度为8的打印出来。

```
5   FOR I=1 TO 10
10    INPUT A$
20    IF LEN (A$) =8 THEN PRINT A$
30  NEXT I
40  END
```

**(2) ASC函数**

形式为：**ASC**（字符串）

得到一个字符串中第一个字符的ASCII码。如：

```
10  PRINT ASC ( "B")
```

```
20   A$= "BOOK"
30   PRINT ASC (A$)
```

"B" 的 ABCII 码为十进制的 66。因此 10 语句和 30 语句都打印它同一个数值 "66"，即

```
RUN
66          66
```

(3) **CHR$函数**

形式为：**CHR$**（算术表达式）

它执行 ASC 函数的反转换，它的值是一个字符，这个字符的 ASCII 码是 CHR$ 函数中括弧内的表达式的值。例如：

```
10   B$=CHR$ (65)
20   C$=GHR$ (33)
30   PRINT B$, C$
```

打印出字符 "A"（代码为 65）和字符 "!"（代码 33）。可以利用此功能打印出一些图示代码。ASCII 码中的 129-191 对应的不是一般字符而是某些特殊的图示符号。有兴趣的读者可以试一下下面语句的打印情况。

```
10 PRINT CHR$ (131)
```

(4) **STR$函数**

它的作用是将一个数值或一个算术表达式的值转换成字符串形式。即将此数值变成用引号括起来的字符串。

一般形式为：**STR$**（算术表达式）

**【例 8.14】**

```
10   A=111
20   B=-222
30   C$=STR$ (A): D$=STR$ (B)
40   PRINT C$
50   PRINT D$
60   END
RUN
 111
-222
```

30 行的作用是将数值 111 转换为 "111"（注意对于正数 A 留一前导空格然后赋给 C$），将-222 转换为 "-222"，然后赋给 D$。

(5) **VAL 函数**

形式为：**VAL（串表达式）**

执行与 STR$函数的相反的功能。将字符串转换为数值。例如：A$ = "105"，B$ = "201"，则，VAL（A$+B$）就得到数值 105201。

如果有 VAL（T$），而 T$中既有数字又有字母或其它字符，则只有最前面的数字有效，其它的字符都被忽略。如 VAL（"10 DOLLARS"）得到 10。VAL（10 DOLLARS AND 50 CENTS"）也得到 10。

**【例 8.15】** 街道上门牌号是单数在路南、偶数在路北。如果我们输入一个地址，希望

由计算机打印出某门牌号在路北还是在路南。

```
10  INPUT "ADDRESS: NUMBER AND STREET"; A$
20  C=INT (VAL (A$) /2) *2
30  IF C=VAL (A$) THEN PRINT "NORTH SIDE" ELSE PRINT "SOUTH SIDE"
100 END
RUN
ADDRESS: NUMBER AND STREET? "1173 BEIJING ROAD" ↙ (输入地址)
SOUTH SIDE
```

程序中 A$ 值为"1173 BEIJING ROAD"，VAL（A$）的值为 1173，由于它已转换成数值，故可以进行算术运算。20 语句中 INT（VAL（A$）/2）*2 的值为：INT（586.5）*2，即 586*2。C 值为 1172。在 30 语句中，由于 C≠VAL（A$），由此可以知道 C 是奇数（如果 C 是偶数，则 C=VAL（A））。故打印出"SOUTH SIDE"。

**（6）STRING$ 函数**

形式为：**STRING$**（n，字符）

可以得到一个由 n 个指定的字符组成的字符串。如：

STRING $（40，"*"）表示 40 个星号。字符也可以用数字代替，此时按 ASCII 码转换为字符，如 STRING$（20，65）得到 20 个"A"字符。

这个函数在制图或打印表格时是有用的，例如打印一行"—"或"*"等。

**（7）INSTR 函数**

形式为 **INSTR**（[n,] 串 1，串 2）

其作用是：从串 1 的第 n 个字符开始，找串 2 在串 1 中第一次出现的位置。串 1 和串 2 可以是字符串，或字符串变量或字符串表达式。n 是算术表达式，如果省略 n，表示从串 1 中的第一个字符开始找串 2。如果 n>LEN（串 1）或串 1 为空串，或串 2 找不到，则 INSTR 函数的值为零。

**【例 8.15】**

```
10  A$=" BASIC PROGRAMMING"
20  PRINT INSTR (A$," BASIC"),
30  PRINT INSTR (A$," A"),
40  PRINT INSTR (5, A$," A"),
50  PRINT INSTR (A$," W"),
60  END
RUN
 1          2          12          0
```

20，30，50 语句中的 INSTR 函数从 A$ 中第一个字符开始搜索，40 语句中的 INSTR 函数从 A$ 中第 5 个字符开始找字符"A"，它的值为 12（而不是 2）。由于 A$ 中无"W"，因此 50 语句输出 0。

**【例 8.16】** 图书查阅。如果你想借一本书，但记不清它的全名，只记得其中一些字，希望计算机能将包含这几个字的书名都打印出来，以便你从中选择。

```
10  INPUT " Enter your word's:",X$
20  FOR I=1 TO 8
30    READ A$
```

```
40    IF INSTR (A$, X$) >0 THEN PRINT A$
50    NEXT I
60    DATA " FORTRAN 77"," BASIC LANGUAGE"," PROBLEM SOLVING WITH BASIC"
70    DATA " PASCAL PROGRAMMING"," ANSI STRUCTURED BASIC"," IBM PC BASIC"
80    DATA " STRUCTURED PROGRAMMING IN BASIC"," LISP"
90    END
RUN
Enter your word' s: BASIC↙
BASIC LANGUAGE
PROBLEM SOLVING WITH BASIC
ANSI STRUCTURED BASIC
IBM PC BASIC
STRUCTURED PROGRAMMING IN BASIC
```

今假设只有 8 本书，将要找的书名中记得的单词输入 X$，然后用一个循环读图书馆中已有的书名，每次读入的书目放在 A$ 中。如果 A$ 中包括 X$，则 INSTR (A$, X$) 必大于 0，此时将 A$ 输出即可。

## 习 题

8.1 写出下面程序运行的结果：

```
10   READ A$, B, C$, D
20   PRINT C$; D, A$; B
30   DATA "BOY", 12, "GIRL", 16
40   END
```

8.2 下面的程序有什么错误。

(1)
```
10 READ A, B$, C, D$
20 PRINT A, B$, C, D$
30 DATA BOOK, 4, PEN, 5
40 END
```

(2)
```
10 A$ =HAPPY
20 PRINT A$
30 END
```

8.3 写出下面程序运行的结果：

```
10 A$ = "THE PEOPLE'S␣"
20 B$ = "␣CHINA"
30 C$ = "REPUBLIC␣"
40 D$ =A$ +C$ + "OF" +B$
50 PRINT D$
60 END
```

8.4 写出下面比较的结果。

(1) "BOOK" 和 "BOY"

(3) "HIT" 和 "BOX"

(5) "1234" 和 "1321"　　(4) "CAT" 和 "BOG"

(7) "B12" 和 "B21"　　(6) "10241" 和 "1024"

(2) "WHAT" 和 "YOUNG"　　(8) "T1B" 和 "123"

8.5 某一学生的每一门课程名称、学时和成绩（分 A、B、C、D 等）已分别表示在 DATA 语句中。现要求单独地把得 D 等（不合格）的课程打印出来。请编出此程序。

```
100  DATA "ENGLISH", 4, "A", "CHINESE", 2, "C"
110  DATA "STAT", 3, "B", "PHYSICS", 4, "D"
120  DATA "PHILOSOPHY", 2, "A", "CALCULUS", 4, "D"
```

8.6 写出下面程序的运行结果。

```
10  A$="$10.85 EACH,THE PROFIT MARGIN IS 0.118"
20  B$=LEFT$(A$,2)+"3.50"+MID$(A$,7,28)+"0.105"
30  PRINT A$
40  PRINT B$
50  END
```

8.7 如已对 A$ 赋值，内容为 "ABCDEFGHIJKLM"。

请写出以下的字符串函数的值：

(1)LEFT$(A$,3)　　(2)RIGHT$(A$,5)

(3)MID$(A$,4,2)　　(4)MID$(A$,6,3)

(5)RIGHT$("ABCDE",3)　　(6)LEFT$("HIJK",3)

8.8 下面是一个查职工工资的程序。请写出当：

(1) 键盘输入的名字为 "CHIN" 时；

(2) 键盘输入的名字为 "SU" 时；

(3) 键盘输入字符串 "END" 时；

程序运行的结果。

```
10   INPUT "TYPE NAME TO FIND PAY"; B$
20   IF B$ = "END" THEN END
30   FOR I=1 TO 6
40     READ A$
50     IF B$=LEFT$ (A$, 4) THEN I=10
60   NEXT I
70   IF I >=10 THEN PRINT "CAN'T FIND NAME" ELSE PRINT LEFT$ (A$, 4); "'S PAY IS";
     MID$(A$, 6, 3); "YUAN"
100  RESTORE
110  PRINT: INPUT "TYPE NAME TO FIND PAY"; B$
120  GOTO 20
130  DATA "FUNG 62", "LING 106", "CHIN 89"
140  DATA "WANG 46", "TAIN 78", "LUNE 69"
150  END
```

8.9 输出一串字符。然后输入一个需要插入的字符，并指定该字符应插入的位置。要求把该字符插入到字符串中指定的位置上；不断重复此过程直到输入指定的位置号为 "-1" 时为止。假设插入字符的总数不超过初始时字符的个数，每次插入一个字符。(提示：可以先把字符串放入一个字符串数组中，在一个数组元素中存放一个字符，然后进行插入。也可以应用

有关的字符串函数和字符串表达式完成插入操作。）

8.10 把一个字符串中的字符拆开，依次逐个放入 X $ 数组的每个数组元素中（即每个中只放一个字符）。

8.11 将本章例8.7（打印熊猫图案）程序改用一维数组去实现。

8.12 将例8.7程序改为不用数组来实现题目要求，用子字符串方法实现。

8.13 自己画出一个图案，用自己编写的程序打印出此图案。

8.14 将两条曲线打印在同一坐标上。

$y1=\sin x$

$y2=\cos(2x+30°)$

要求 $x$ 轴在第 40 列，在 40 列打印一条虚线（以表示轴线），曲线最大振幅为 40，每 10°打印一个点。y1 用“*”表示，y2 用“0”表示。要求用字符串数组实现。

8.15 将上面的打印结果转 90°，即 $y$ 轴与打印机走纸方向一致，$x$ 轴与行的方向一致（与人们习惯的坐标一致）

8.16 从一个给定的字符串中删去和增加某些字符，如给定 A $ = “I have a computer”。①先把“a”删去；②再在“a”原来位置上加入“three”；③在“computer”后加入二个字符“s”和“.”。

8.17 译密码。为保密，常将电报改用密码输出，今规则如下：将“A”改为“F”，“B”改为“G”，即将一个字母改为它后面第 5 个字母。对最后 5 个字母的规则为：“V”⇒“A”，“W”⇒“B”⇒，“X”⇒“C”，“Y”⇒“D”，“Z”⇒“E”。如“BASIC”的密码应为“GFXNH”。编一程序将输入的一行字符改用密码输出（只改字母，对非字母字符按原样输出）。

# 第九章 屏幕控制和作图

BASIC 除了提供数值运算和字符处理的功能以外，还提供屏幕控制功能和作图功能，允许用户方便地控制在屏幕的不同位置上显示字符或图形。可以利用屏幕控制功能设计“菜单”（见第十章），以加强与用户的“友好性”。利用图形功能可以用曲线、竖方图，图形形式输出各种信息，以增加形象性。许多领域（例如计算机游戏、计算机辅助设计、计算机辅助教学等）都需要用到图形功能。

图形或图象是一种能迅速且精确地传递信息的有效方法，容易被人们理解和记忆。利用计算机来绘制各种图形、图表已成为各个领域的重要手段，例如，计算机辅助设计和辅助制造（CAD/CAM）、计算机模拟、图象处理、地学信息处理及地图绘制、企业管理、办公室自动化以及计算机辅助教学（CAI）等。

IBM－PC 机提供两种屏幕显示方式，即文字显示方式和图形显示方式，或称字符显示模式和图形显示模示。在文字显示模式下，显示的最小单位是字符，用户只能用字符和专用线条来组合图形，显然是很粗糙的。在图形显示模式下，显示的最小单位是显象管的扫描点，并且可以指定任何一个点的显示颜色。因而，通过精心设计能获得复杂而美观的彩色或黑白图案。

IBM－PC 机有的配置了彩色/图形适配器，给绘制彩色图形提供了很好的条件，本章的所有例题都是在此条件下做的。但是，大部分例题（除例 9.3 和例 9.5 外）也可以在只配黑白图形适配器的 PC 机上运行，为了清楚地显示出黑白图形，可将程序中彩色语句去掉和留意一下显示器的纵横比就可以了。所以，这一章的内容对那些只备有单色图形适配器 PC 机的读者也是很有用的。为了了解计算机绘图的基本原理，先介绍计算机屏幕的控制。

## §9.1 屏幕控制语句

### 9.1.1 LOCATE 语句和 WIDTH 语句

当你起动 BASIC 编辑一个源程序时，屏幕显示的状态就是字符显示模式（有时也称文本方式）。可以输入源程序或进行编辑、修改、运行源程序的各种操作。屏幕在此模式下，可以显示字符 25 行，每行 40 个字符或 80 个字符，这可以由宽度语句 WIDTH 来决定，它的格式是：

**WIDTH n**

其中，**n** 的数值可以是 40 或 80。

若 **n** 是 80，那么屏幕每行显示 80 个字符，一个屏幕最多给出 25×80（2000）个字

符位置，这些不同的字符位置将屏幕分成一个个小小的矩形，这些小矩形是按行和列排列的。我们把行编号成1到25，行1在屏幕的最上部（顶部）；行25在屏幕的最下部（底部）（这是一般 **BASIC** 用来显示功能键的地方）。列编号为1到80，最左边的为列1，最右边的为列80，这样就可以对屏幕上的每一个小矩形的位置确定下来，其坐标如图9.1所示。

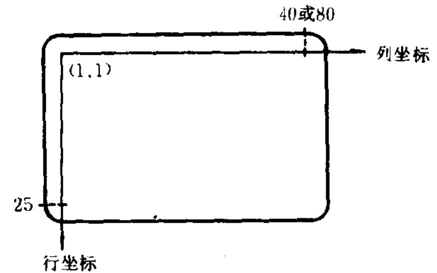


图9.1

我们可以用定位语句 **LOCATE** 来确定屏幕光标的位置和属性。它的格式为：

**LOCATE** [〈**行**〉] [，[〈**列**〉] [，[，〈**方式**〉] [，[〈**始线**〉] [，[〈**末线**〉]]]]

语句后面这些参数的有效值和含义请见表9.1。

**表 9.1**

| 参　数 | 有　效　值 | 说　　明 |
|---|---|---|
| 行 | 1～25 | 移动光标到所指定的行数 |
| 列 | 1～40或1～80 | 移动光标到所指定的列数 |
| 方式 | 0或1 | 0熄灭光标；1点亮光标 |
| 始线 | 0～7或0～13 | 光标的开始扫描线 |
| 末线 | 0～7或0～13 | 光标的结束扫描线 |

其中，“行和列”可以用算术表达式表示，取值如表9.1所示，用来确定字符显示方式下的光标位置；“方式”用来确定光标是否可见，取0表示不可见，取1表示可见；“始线和末线”也可以算术表达式，其值用来确定光标面积（或高度）的大小。

通常 BASIC 下的光标是一根扫描线，其实屏幕上的每个字符可以由8根（0～7）扫描线（具有彩色图形适配器的 PC 机）组成，也可以由14根（0～13）扫描线（具有黑白图形适配器）组成。线0在一个字符的顶部。例如：

LOCATE,,，0，7

将产生一个大的长方形光标，占据整个字符位置上。一般来说，需要用多参数的图形控制语句，但你可以省略一些不需要改变的参数，上面这句就是一个例子，省略了“行”，“列”和“方式”。也就是光标的位置和方式照原来的不变，只改变了光标的高度（或面积）的大小。LOCATE 语句不会影响在屏幕上已经出现的字符，如果移动光标到已经出现的字符上，那么光标只是在该字符上闪烁而已，不会改变那个字符。

当你用 LOCATE 语句定位了光标位置，你就可以用 PRINT 语句在指定的位置上显示字母，数字或其他一些专用的字符。

### 9.1.2 CSRLIN 和 POS 函数

这两个函数，某种意义上来说，跟 LOCATE 语句相反，它们不是移动光标到指定的位置，而是把当前光标所在的行和列保存起来。例如你要求一个程序在执行输出的过程中在 25 行上输出一个信息以后再在屏幕输出的原来地方（即显示 25 行信息之前的地方）继续输出，可用如下程序段：

```
100  ROW=CSRLIN
110  COL=POS (0)
120  LOCATE 25, 1
130  PRINT MESSAGE
140  LOCATE ROW, COL
```

CSRLIN 函数记录了光标所在行的位置，而 POS 函数记录了光标所在列的位置，它们的值分别赋值给变量 ROW 和 COL，而后定义光标位置到 25 行的第 1 列，输出信息（MESSAGE）后，140 语句的 LOCATE 使光标返回到原来的地方。

### 9.1.3 KEY OFF 和 KEY ON 语句

KEY OFF 语句用来关闭第 25 行功能键的显示，以便整个屏幕用来绘图或输出其他信息。需要重新恢复功能键的显示时用 KEY ON 语句。

### 9.1.4 CLS 语句

它是用来清除屏幕并置光标于屏幕的左上角（字符显示模式）或置屏幕的中心（图形显示模式）。

下面介绍，应用以上语句的两个程序。

**【例 9.1】** 在计算机屏幕上显示一个表格，并要求在计算机上填写此表格。这样，光标就好象是“一支笔”，用户只要通过键盘输入填写，每填空一项，按回车键以后光标就应该自动“跳”到下一项的开始位置，一直到全部项目填写完毕。例中要求屏幕在适当的位置显示如下表格和填写 7 项内容（名字、性别、年龄、地址、城市名、邮政编码和电话号码）。

Name：________________ Sex：________ Age：________

Address：________________________

City：________________ Zip - code：____________

Phone：__________

程序和运行结果如下：

```
10   REN Using LOCATE to create a Screen Format
100  KEY OFF: CLS; LOCATE,,, 0, 7
110  READ NFIELD
120  FOR I=1 TO NEIELD
130    READ LIN, COLA, LABEL$, TEXT$ COLB
```

```
140     LOCATE LIN, COLA
150     PRINT LABEL $ ; TEXT $ ;
160   NEXT I
170   RESTORE: READ NFIELD
180   FOR I=1 TO NFIELD
190     READ LIN, COLA, LABEL $ , TEXT $ , COLB
200     LOCATE LIN, COLB
210     INPUT"", RESP $
220   NEXT I
230   LOCATE ,,, 7, 7: KEY ON
1000   DATA 7
1010   DATA 5, 3, "Name:", "______________________", 9
1020   DATA 5, 27, "Sex:", "_________", 32
1030   DATA 5, 38, "Age:", "_______", 43
1040   DATA 7, 3, "Address:", "____________________________", 12
1050   DATA 9, 3, "City:", "__________________", 9
1060   DATA 9, 28, "Zip-code:", "____________", 38
1070   DATA 11, 3, "Phone:", "__________", 10
1080   END
RUN
    Name: _ Bu jia-qi__________Sex: _M_____Age: _48___
    Address: _149# Yang Chang Road______
    City: _Shanghai________Zip-code: _200072_____
    Phone: __625744__
OK
```

其中，变量 NFIELD 代表填写表格的总项数；LIN 代表行；COLA 代表列；LABEL $ 代表项目名称；TEXT $ 代表填写内容的下划线；COLB 是填写项光标的起始位置；RESP $ 代表从键盘输入的填写内容。

100 语句的作用是取消屏幕第 25 行的功能键显示、清屏、并设置一个长方形的光标；110～160语句在屏幕上制作表格；语句 180～220 从键盘输入填写表格；语句 230 恢复光标成一条扫描线和第 25 行的功能键显示；语句 1000～1070 为用 DATA 语句提供的工作数据，改变其值，就可以得到不同的表格形式。

**【例 9.2】** 在屏幕上画一个坐标，在垂直轴上标以 Profit，在水平轴上标以 Month。程序如下：

```
10   REM Using LOCATE to create COORDINATES
20   CLS: WIDTH 80: LOCATE 1, 1
30   PRINT "Profit"
40   LOCATE 6, 40
50   PRINT "Month"
60   FOR J=1 TO 4
70     LOCATE j, 7: PRINT CHR $ (179);
80   NEXT J
```

```
90  LOCATE 5, 7: PRINT CHR$ (192);
100 FOR J=8 TO 40
110   LOCATE 5, J: PRINT CHR$ (196);
120 NEXT J
130 END
```

运行结果如图 9.2 所示。CHR $ (179) 是一条短直线"|" CHR $ (192) 是"└", CHR $ (196) 是横短线"—"。

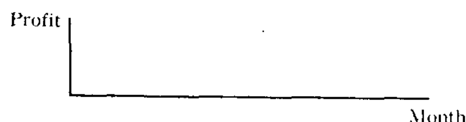



图 9.2

以上两例都是在字符显示模式下进行的，这是在引入 BASIC 时就设置好的。字符显示模式下，能显示各种 ASCII 码字符及一些专用图形符号，共 254 个（见附录 I）。

但怎样从字符显示模式转入图形显示模式（或反之）？这要用到 SCREEN 语句。

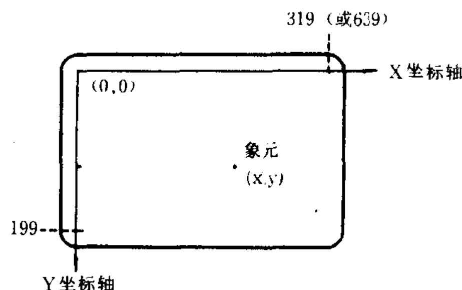



图 9.3

## 9.1.5 SCREEN 语句

文本方式能控制字符的工作模式，而图形显示模式（或简称图形方式）能控制扫描点（也称象素或象元）的工作方式。一幅图画，实际上是由成千上万个象元组成的，所以屏幕上能够显示出来的象元个数，就决定成象的质量。在图形模式下，又有中分辨率和高分辨率之分。中分辨率工作模式时，每屏可显示 200 条线，每一条线则有 320 个点，所以中分辨率模式时的全屏象元个数为 200×320 个。高分辨率时，每屏也是显示 200 条线，但每一条线有 640 个点组成，这时全屏象元总数是 200×640 个。此外，中分辨率时，每个象元可以有 4 种不同的颜色，而高分辨率时，每个象只是有黑白两色。屏幕上的每个象元都有一个坐标位置，所在行是用编号 0～199，列的编号用 0～319（中分辨率）或 0～639（高分辨率），其 0 点坐标如图 9.3 所示。

SCREEN 语句的简化格式是：

**SCREEN [〈模式〉], [,〈彩色〉]**

其中模式为显示器工作的选择参数，为整型表达式，其值为：

0　选择字符显示模式；

1　选择中分辨率图形显示模式；

2　选择高分辨率图形显示模式。

彩色用来选定是黑白或彩色显示，规定其值为：

0 { 字符显示（即模式为 0）时，黑白显示。
    中分辨率图形（即模式为 1）时，彩色显示。

非 0 { 字符显示（即模式为 0）时，彩色显示。
       中分辨率图形（即模式为 1）时，黑白显示。

当高分辨率显时（即模式为 2），不管彩色为何值，都是黑白显示（见表 9.2）。

表 9.2

| 模式参数 \ 彩色参数 | 0 | 非 0 |
|---|---|---|
| 0（字符方式） | 黑白字符 | 彩色字符 |
| 1（中分辨） | 彩色图形 | 黑白图形 |
| 2（高分辨） | 黑白图形 | 黑白图形 |

SCREEN 语句的所有参数均可省略，缺省参数保持已经执行过的当前值。例：

```
10   screen 0，1
20   screen 1
30   screen ，0
40   screen 2
```

例中 10 语句，选定模式为 0，彩色为 1，因此是字符彩色显示模式。20 行语句中选择模式为 1，彩色缺省，因此仍保持原来彩色为 1 的值，这时显示器为中分辨率黑白显示模式。30 语句保持模式为 1，选择彩色为 0，因此显示器成为中分辨率，彩色图形显示模式。40 语句选择模式为 2，置屏幕为高分辨率显示模式，所以不论彩色的值如何（此时为 0），均为黑白显示。顺便提一下，在图形模式下，也可用 WIDTH 语句，此时 WIDTH 40 相当于 SCREEN 1，WIDTH 80 相当于 SCREEN 2。

以上选择，只是使显示器处于某种工作模式。至于屏幕上显示出什么颜色的图形和背景（底色），还要由 COLOR 语句配合使用。而 COLOR 语句在字符显示模式下和图形显示模式下有不同的含义，所以下面分开叙述。

### 9.1.6 字符显示模式下的 COLOR 语句

它的格式是：

**COLOR** [〈**字符颜色**〉] [，〈**背景颜色**〉]

其中；字符颜色是算术表达式，其值定义字符的显示颜色，取值为 0～31，具体规定为表 9.3 所示。

表 9.3

| 色 号 | 色 名 | 色 号 | 色 名 |
|---|---|---|---|
| 0 | 黑 | 8 | 灰 |
| 1 | 蓝 | 9 | 浅蓝 |
| 2 | 绿 | 10 | 浅绿 |
| 3 | 青 | 11 | 浅青 |
| 4 | 红 | 12 | 浅红 |
| 5 | 洋红 | 13 | 浅洋红 |
| 6 | 棕 | 14 | 黄 |
| 7 | 白 | 15 | 高亮度白 |

16～31 在原颜色 0～15 的基础上加 16，定义字符的彩色为闪烁方式。

背景颜色也是算术表达式，其值定义屏幕的底色，取值范围是 0 到 7。具体规定为表 9.4 所示。

表 9.4

| 色　号 | 色　名 | 色　号 | 色　名 |
|---|---|---|---|
| 0 | 黑色 | 4 | 红色 |
| 1 | 蓝色 | 5 | 洋红 |
| 2 | 绿色 | 6 | 棕色 |
| 3 | 青色 | 7 | 淡灰 |

【例 9.3】　在彩色图形适配器的 PC 机上显示 16 种不同的字符颜色和 8 种不同的背景颜色。程序中显示的字符是：

* * * * * Good Morning * * * * *

程序如下：

```
10   REN Using COLOR (Text Mode)
20   SCREEN 0, 1
30   FOR I=0 TO 15
40     FOR J=0 TO 7
50       COLOR I, J
60       PRINT "* * * * * * * * * * * Good Morning * * * * * * * * * * *"
70       FOR K=1 TO 100 : NEXT K
80     NEXT J
85     STOP
90   NEXT I
100  SCREEN 0, 0
110  END
```

20 行语句设置屏幕为字符显示模式，彩色显示；30～90 行语句显示不同背景下的 16 种不同颜色的字符；而 85 行 STOP 语句用来暂停，便于观察，继续运行按 F5。

当用只具有单色（黑白）图形适配器时，利用 COLOR 语句不能产生彩色，但可以产生闪烁、负象、消隐以及加上下划线等不同的字符显示。字符颜色取值也是 0～31，但其中

0　　黑色

1　　加下划线的白色字符

2～7 白色

上述数字（0～7）加上 8 得到相应的高亮度色，如 9 为给出带下划线的高亮度白色字符，15 给出高亮度白色，但不能给出高亮度黑色。上述数字（0～15）加上 16，定义字符闪烁。背景颜色取值 0～7，其中 0～6 为黑色，7 为白色。例 9.4 是字符加亮，闪烁、负象，消隐的实际应用。

【例 9.4】　核对口令

有些计算机，用户在开机以后，要求核对预先已告许计算机的口令（本题假定你的口令是“GOOD”或“good”，其程序段如下：

```
100   REM Check password
110   CLS: PRINT: PRINT
130   COLOR 7, 0: PRINT "enter your"
140   COLOR 15, 0: PRINT "PRASSWORD"
150   COLOR 7, 0: PRINT "here:"
160   COLOR 0, 0:
170   INPUT PASSWORD$
180   WHILE PASSWORD$<> "GOOD" AND PASSWORD$<> "good"
190     I=I+1
200     CLS
210     IF I=5 THEN COLOR 0, 7: PRINT "you are illegal user!": COLOR 7, 0: END
220     COLOR 31, 0: PRINT "Wrong Password"
230     COLOR 7, 0: PRINT "Enter your PASSWORD again...."
240     COLOR 0, 0: LOCATE 3, 24
250     INPUTT PASSWORD$
260   WEND
270   CLS: COLOR 0, 7
280   PRINT "Program continue....."
290   COLOR 7, 0
```

程序中，没有 SCREEN 语句，那么就认为是 SCREEN0，0，即字符显示模式，且是黑白的。110 行语句清屏；130 行语句中 COLOR 语句的字符颜色是 7（白色），背景颜色是 0（黑色），所以此时的字符显示是普通的字符显示，而 140 行语句中的 COLOR 语句，其字符颜色是 15（即 7+8），是高亮度白色，所以这一行显示出的字符 PASSWORD 是高亮度白色，以引起用户的注意，150 行语句又恢复普通显示。所以这三行在屏幕上输出如下内容：

enter your

PASSWORD　　　（此行高亮度）

here：

?

160 行的 COLOR 语句中，其字符颜色和背景颜色都是 0，即黑色，所以此后输入的字符（或输出的字符）在屏幕上是看不出来的，170 行语句要求从键盘输入口令，一般这个口令是保密的，只有用户自己知道，此时，你对你自己的口令要熟记，千万不要记错了，因为记错了就"进不去"，输出的口令在屏幕上不显示，若输入口令跟预约的口令不符，那么程序运行到 220 行语句，此时 COLOR 语句中的字符颜色是 31（即 15+16），是加亮白色再加闪烁的显示，所以输出的"Wrong Password"是加亮白色且加闪烁，以示警告，接着又用普通显示要求你重新输入口令的字样出现在下一行，这二行屏幕上输出内容是：

Wrong Password

Enter your PASSWORD again.....

因为，此时又要求作输入口令，那么又要通过 COLOR 语句设置输入字符是不可见的，所以程序运行到 240 行语句，其中的 COLOR 语句的字符颜色和背景颜色又都是 0，表示输入字符不显示，若你的口令连续 5 次输错，那么程序运行至 210 行语句，此时 COLOR 语句的字符颜色是 0（黑色），而背景颜色是 7（白色），因此出现负象（反白）输出：

you are illegal user!　　　(非法用户)

Ok

后面再用 COLOR 语句恢复普通显示，程序运行结束。这样你就没有“进入”，假定在 5 次中，有一次是正确的，那么程序执行 270 行语句，以负象形式显示出：

Program continue.....

290 行恢复普通显示，程序继续执行，表明你已“进入”。

### 9.1.7 图形显示模式下的 COLOR 语句

在中分辨率图形显示模式下，屏幕上的每个点，都可以有多种颜色，COLOR 语句在图形显示模式下的格式为：

**COLOR [〈背景颜色〉] [,〈配色器号〉]**

其中参数“背景颜色”是一整型表达式，取值 0～15 之间，其数值所对应的颜色跟字符显示模式下的字符颜色相同。配色器号用以选择配色器是 0 或 1，为整型表达式。当取偶数时，选定 0 号配色器；当取奇数时，选定 1 号配色器。在中分辨率图形显示模式时，每个象元可以有 4 种不同的彩色码，即 0，1，2 和 3，它们所显示的颜色，由程序中 COLOR 语句选定的配色器决定，具体规定为表 9.5 所示。

表 9.5

| 彩　色　码 | 0 号配色器 | 1 号配色器 |
|---|---|---|
| 0 | 与底色相同 | 与底色相同 |
| 1 | 绿 | 青 |
| 2 | 红 | 洋红 |
| 3 | 黄 | 白 |

**【例 9.5】** 在中分辨率图形模式下显示 16 种不同的屏幕底色，程序如下：

```
5 Draw 104 Using COLOR (Graphical Mode)
10   SCREEN 1, 0
20   FOR I=0 TO 15
30     COLOR I, 1
40     FOR J=1 TO 1000: NEXT J
50   NEXT I
60   SCREEN 2
70   END
```

此例中，10 行为设置屏幕为中分辨率彩色显示模式；20～50 行使屏幕底色由黑、蓝……一直变到白色（亮白）；40 行语句起延时作用，以便在底色变换时停留片刻，便于观察。

## §9.2 画点和画线

### 9.2.1 画点语句

MS BASIC 在显示器处于图形模式下，有两条画点语句，即 PSET 和 PRESET 语句。

(1) PSET 语句

它的是格式是：

**PSET** ($x,y$)，[，<彩色>]

其功能是“点亮”($x,y$)坐标点。$x$ 和 $y$ 是算术表达式，其取值范围不可以超过屏幕坐标系所允许的值，也就是说：

中分辨率时　$0\leqslant x\leqslant 319$　$0\leqslant y\leqslant 199$

高分辨率时　$0\leqslant x\leqslant 639$　$0\leqslant y\leqslant 199$

“彩色”是该点象元的彩色代码，也就是我们在第一节里所提到的（图形显示模式下的 COLOR 语句中）那个彩色码。当显示器处于中分辨率时，取值为 0 到 3；高分辨率时，取值 0 或 1。其内定值（即缺省此项时）中分辨率是 3，高分辨率是 1。但是，被点亮的象元究竟是什么颜色，还与 COLOR 语句中配色器的选定有关，请查阅上一节的图形显示模式下的 COLOR 语句以及表 9.5。

(2) PRESET 语句

它的格式是：

**PRESET** ($x,y$) [，<彩色>]

其功能是“擦”去($x,y$)坐标点。实际上它是用屏幕底色画点，语句中各参数的含义同 PSET 语句，彩色缺省时其值为 0，也就是说与底色相同。

**【例 9.6】**　以屏幕模拟天空，画一幅“群星闪烁和流星”。

```
110  REM Using PSET and PRESET
120  KEY OFF : SCREEN 2 : CLS
130  FOR I=1 TO 100
140    X=640*RND : Y=200*RND
150    PSET (X, Y)
160  NEXT I
270  J=200*RND
180  FOR I=200 TO 400
190    PSET (I, J)
200  NEXT I
205  FOR I=1 TO 150 : NEXT I
210  FOR I=200 TO 400
220    PRESET (I, J)
230  NEXT I
240  IF INKEY$<>"" THEN END
250  GOTO 120
```

图 9.4

本程序演示画点语句的应用，120 行语句去掉 25 行功能键的显示，设置高分辨率图形模式和清屏；130～160行语句通过随机函数来决定屏幕坐标（x，y），而后用 PSET 语句画点来模拟群星；170—230 行语句是通过随机地在水平方向利用 PSET 语句画线，经过瞬间延时后再用 PRESET 语句擦去这条线来模拟天空中的 流星。闪 烁是通过不断清屏又再现图形来模拟的。图 9.4 是程序运行的某一瞬间。

### 9.2.2 画线语句

画线语句可以在屏幕上画直线和矩形框，有三种格式，它们是：

(1) LINE　$(x_1, y_1)$ — $(x_2, y_2)$ [，<彩色>]

(2) LINE　$(x_1, y_1)$ — $(x_2, y_2)$ [，<彩色>]，B

(3) LINE　$(x_1, y_1)$ — $(x_2, y_2)$ [，<彩色>]，BF

其中，坐标 $(x_1, y_1)$ 为起始点，坐标 $(x_2, y_2)$ 为终止点，格式（1）的含义是从坐标点 $(x_1, y_1)$ 开始到坐标点 $(x_2, y_2)$ 画一根直线，此直线的颜色由彩色参数（即彩色码（0～3）和相应的 COLOR 语句中的配色器号所决定。也就是说彩色这个参数跟画点语句 PSET 中的规定一样，缺省值是 3。

格式（2）比格式（1）多一了个参数 B，B 是一个标识字符，它的作用是使语句由画一条线变成画一个由坐标 $(x_1, y_2)$ 和 $(x_2, y_2)$ 的连线为对角线的矩形。

格式（3）基本上跟格式（2）相同，但是此时的标识字符是 BF，其功能不仅是画一矩形，而且在矩形内填上跟矩形边框一样的颜色。

三个语句中的 $(x_1, y_1)$ 坐标点可以缺省不写，此时表示起始点为当前点，也就是说是前面一个画图语句的终点。

**【例 9.7】** 在绿色屏幕上画一根黄线、而后画一个由黄线组成的矩形，接着再画一个由黄色填满的矩形。程序如下：

```
5  REM Using LINE Statement
10  SCREEN 1, 0 : CLS
20  COLOR 10, 0
30  LINE (30, 30) — (50, 50)
40  LINE (50, 50) —STEP (10, 10), 3, B
50  LINE—(100,100),,BF
60  IF INKEY $ = "" THEN 60
70  SCREEN 0
80  WIDTH 80
90  END
```

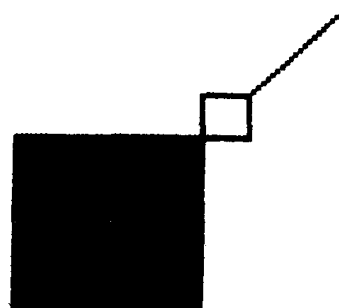


图 9.5

程序中，10 行语句设置屏幕为中分辨率彩色显示；20 行语句设置屏幕底色为浅绿和配色器号为 0；30—50 行语句的彩色参数均为 3，由表9.5中可以查到此时绘图颜色是黄色；70～80 行语句为恢复屏幕为字符显示，且行宽是 80 个字符。

程序中的 STEP（10，10），可以用坐标（60，60）代替，这里顺便说明一下，坐标（x，y）有两种使用形式，一种形式是绝对坐标，即取相对于坐标原点（0，0）的值，上面我们在

画点语句和画线语句中使用的是绝对坐标的形式。也可以是另一种坐标形式——相对坐标形式，取相对于当前点的坐标（位移量）。这时语句中的坐标点用 STEP（Dx，Dy）表示，其中（Dx，Dy）是相对于当前点的坐标。

程序运行结果如图9.5所示。如果此程序在具有单色图形适配器的计算机上运行，则须在20行语句前加一个撇号（把它变成注释语句），此时图形是黑白的。

**【例 9.8】** 在屏幕上画一个方框校验板，颜色是红黑间隔，其中小方格的尺寸可由键盘输入。程序设计步骤是：

(1)选择中分辨率图形模式；

(2)设置底色为黑色，配色器为 0 号(因为用 0 号配色器时，用彩色码 2 即可得到红色)；

(3)读方格校验板的左上角坐标，(设为 XSTART，YSTART)

(4)输入小方格的尺寸；

(5)画方格校验板。

```
100  REM Using LINE to create a CHECKBOARD
110  KEY OFF : SCREEN 1 : COLOR 0, 0
120  READ XSTART, YSTART
130  INPUT "hight of squares (number of points)" : N
140  CLS
150  XSTEP=N*1.2 : YSTEP=N
160  Y1=YSTART
170  RED=-1
180  FOR ROW=1 TO 8
190    X1=XSTART
200    FOR COL=1 T) 8
210    IF RED=-1 THEN LINE (X1, Y1) -STEP (XSTEP, YSTEP), 2, BF : RED%=0
       ELSE LINE (X1, Y1) -STEP (XSTEP, YSTEP), 2, B : RED%=-1
220    X1=X1+XSTEP
230    NEXT COL
240    Y1=Y1+YSTEP
250    IF RED=-1 THEN RED=0 ELSE RED%=-1
260  NEXT ROW
270  LOCATE 25
280  INPUT "Carriage returt to end"; X$
290  SCREEN 0 : WIDTH 80 : KEY ON
300  DATA 80, 10
10000  END
```

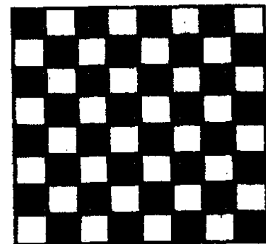


图 9.6

其中 150 语句决定 X 和 Y 坐标的步长值，$y$ 方向的步长 YSTEP 等于输入的小方块尺寸，而 $x$ 方向的步长 XSTEP 要乘以一个系数才能得到与 YSTEP 相同的值，这是由于屏幕有个纵横比的问题，这在画圆语句中将作进一步的解释。程序中心是 210 语句，通过变量 RED 的值为 0 和 −1 的变化来分别执行两个 LINE 语句，这两个语句都是画方框的，不同之处在于一个涂色，一个不涂色。

运行结果图 9.6 所示（小方块尺寸为 10）。

**【例 9.9】** 画函数图形

我们可以在计算机屏幕上画各种各样的函数图形，此例以正弦函数为例，但是程序具有普遍意义，只要稍作改变，就可以画其他函数图形，例9.10就是一个典型的例子。

大家知道，函数值有大有小，那么怎样把一个函数曲线合适地在屏幕上显示出来呢？所谓合适，简单地说就是充分利用屏幕，图形看上去清楚，大小合理。那么，首先要知道的是 $x$ 和 $y$ 的最小、最大值（设为 minx，maxx 和 miny maxy），这在函数分析中一般是可以得到的，而后在屏幕上确定极值坐标，此程序中，屏幕上的极值坐标对应于 x 和 y 极值之间的关系如下：

$$\text{XLEFT} \xrightarrow{\text{对应于}} \text{minx}$$

$$\text{XRIGHT} \xrightarrow{\text{对应于}} \text{maxx}$$

$$\text{YTOP} \xrightarrow{\text{对应于}} \text{maxy}$$

$$\text{YBOT} \xrightarrow{\text{对应于}} \text{minxy}$$

接下来的问题是找极值中间的这些点，也就是要找 x 和 y 值和屏幕上相应坐标点的关系。首先定义数值与屏幕坐标点之间的比率为：

$$\text{SLOPEX} = (\text{XRIGHT} - \text{XLEFT}) / (\text{maxx} - \text{minx})$$

$$\text{SLOPEY} = (\text{YTOP} - \text{YBOT}) / (\text{maxy} - \text{miny})$$

于是，任意 $x$、$y$ 值与屏幕坐标 $x1$，$y1$ 有如下关系：

$$\text{X1} = \text{XLEFT} + \text{SLOPX} * (\text{X} - \text{minx})$$

$$\text{Y1} = \text{YBOT} + \text{SLOPY} * (\text{Y} - \text{miny})$$

这样对应于函数的每个（x，y）值，就可找到相应在屏幕上的坐标（X1，Y1）。再用 PSET（X1，Y1）画点。程序如下：

```
100 REM Drawing a Function Curve
110 SCREEN 1 : KEY OFF
120 READ  STEPX
130 READ MINX, MAXX, XLEFT, XRIGHT
140 READ MINY, MAXY, YBOT, YTOP
150 SLOPEX= (XRIGHT-XLEFT) / (MAXX-MINX)
160 SLOPEY= (YTOP-YBOT) / (MAXX-MINY)
170 D=3.14259/180
180 YBAR= (YBOT+YTOP) /2
190 LINE (LXEFT, YBAR) - (XRIGHT, YBAR)
200 FOR X=MINX  TO  MAXX  STEP  STEPX
210   Y=SIN(X*D)
220   X1=XLEFT+SLOFEX*(X-MINX)
230   Y1=YBOT+SLOPEY*(Y-MINY)
240   PSET (X1, Y1)
250 NEXT X
260 LOCATE 25 : INPUT "carriage return to end"; X$
270 SCREEN 0 : WIDTH 80 : KEY ON
280 DATA 5
```

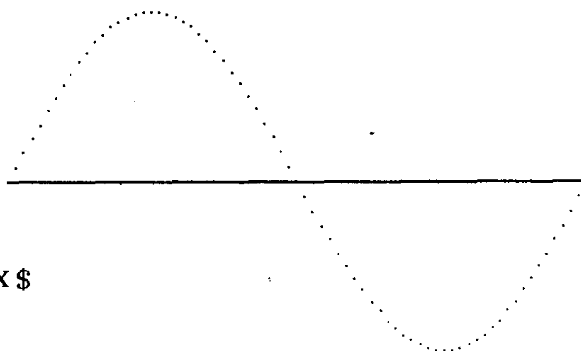


图 9.7

```
290   DATA 0, 360, 10, 310
300   DATA -1, 1, 180, 20
10000   END
```

图 9.7 是程序运行结果。

**【例 9.10】** 画函数 $y = H\sin(x + d)e^{-kx}$ 曲线，其中 $d$ 是初相角（为 90 度），K 为衰减系数取0.05，H 为最大振幅 50。要求连续画出 10 个周期的衰减波形。程序与 9.9 基本相同，改动很少，

```
100   REM Drawing y=h * sin (x+d) * exp (-k * x)
110   SCREEN 2 : KEY OFF
120   READ STEPX
130   READ MINX, MAXX, XLEFT, XRIGHT
140   READ MINY, MAXX, YBOT, YTOP
145   READ B, K
150   SLOPEX= (XRIGHT-XLEFT) / (MAXX-MINX)
160   SLOPEY= (YTOP-YBOT) / (MAXY-MINY)
170   D=3.14159/180
180   YBAR= (YBOT+YTOP) /2
190   LINE (XLEFT, YBAR) - (XRIGHT, YBAR)
200   FOR X=MINX TO MAXX STEP STEPX
210     Y=MAXY * SIN ( (X+B) * D) * EXP (-K * (X * D))
220     X1=LXEFT+SLOPEX * (X-MINX) 230     Y1=YBOT+SLOPEY * (Y-MINY)
240     PSET (X1, Y1)
250   NEXT X
260   LOCATE 25 : INPUT "carriage return to end";
      X$
270   SCREEN 0 : WIDTH 80 : KEY ON
280   DATA 2
290   DATA 0, 3600, 10, 630
300   DATA -50, 50, 180, 20
310   DATA 90, 05
10000   END
```

图 9.8 是这个程序运行的结果。

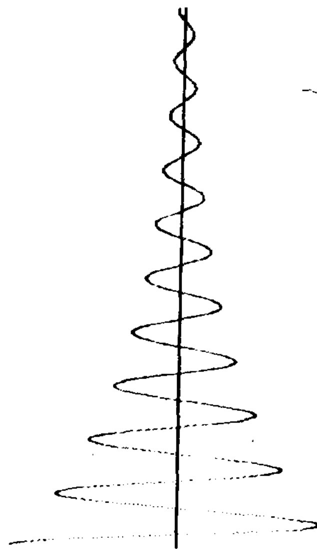


图 9.8

### 9.2.3 DRAW 语句

当一幅画包含很多直线段时，那么要用到很多画线语句，为了更方便地用来连续画各种直线，MS BASIC 提供了 DRAW 语句，它的功能较强，其格式是：

**DRAW<字符串>**

其中，字符串由一系列的画图命令组成，主要包括：

Un——向上移动 n 个单位

Dn——向下移动 n 个单位

Ln——向左移动 n 个单位

Rn——向右移动 n 个单位

En——向右向上（对角线）各移动 n 个单位

Fn——向右向下（对角线）各移动 n 个单位

Gn——向左向下（对角线）各移动 n 个单位

Hn——向左向上（对角线）各移动 n 个单位

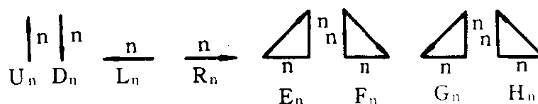



图 9.9

这些命令的作用如图 9.9 所示。

其它还有：

Cn——选择颜色 n，中分辨率 0～3，高分辨率 0～1

Sn——比例因子。n 为 0～225，n 被 4 除是真正的比例因子。

An——把图形旋转一个角度，而这个角度是 90 度的倍数，n 可以是 0、1、2、3，分别对应于把图形旋转 0 度，90 度，270 度。旋转的方向都是顺时针。

Mx,y——从现行位置开始向（x，y）画线。

要深入掌握 DRAW 语句，可参阅有关手册，这里只作基本的介绍，例9.11是应用的一个例子。

**【9.11】** 画一个帆船，背景淡蓝，帆是白色，而船是洋红的。程序如下：

```
5  REM Using DRAW Statement
10  SCREEN 1, 0 : CLS: KEY OFF
20  COLOR 9, 1
25  A $ = "C3 L40 U80 F40 D40 L20 U80.D90 C2 L130 F20 R50 E20 L70"
30  DRAW  A $
35  IF INKEY $ = "" THEN GOTO 35
40  SCREEN 0 : WIDTH 80 : KEY ON
50  END
```

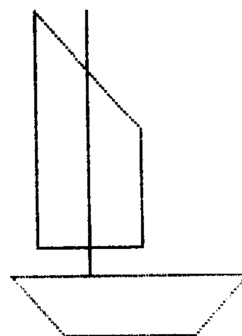


图 9.10

程序中，先定义字符串 A $，而后 DRAW A $，当然也可以把 A $ 右边的表达式直接放在 DRAW 的后面，其效果是一样的。程序运行结果图9.10所示。

## §9.3 画圆、椭圆和圆弧

圆、椭圆及圆弧都可以用 CIRCLE 语句来完成。它的一般格式是：

**CIRCLE($x_c$,$y_c$),<半径>[,<彩色>[,<起始角>,<终止角>[,<纵横比>]]]**

CIRCLE 语句的作用是在显示屏上以（$x_c$,$y_c$）为圆心，r 为半径画一个椭圆。当“纵横比”合适时就成为一个圆，根据起始角和终止角可以指定只画椭圆的一部分，即圆弧。

其中（$x_c$,$y_c$）是椭圆中心点坐标，可以取绝对坐标，也可取相对坐标，相对坐标的用法跟画线语句相同。

“半径”为所画椭圆的半径，即 x 轴的半径（主轴）。“彩色”为画圆弧的彩色码，取值是 0～3，缺省时为 3。“起始角”为所画弧的起始角，以弧度表示，取值为$-2\pi \sim 2\pi$，逆时针度量，缺省时其值为 0。“终止角”为所画圆弧的终止角，以弧度表示，取值为$-2\pi \sim 2\pi$，逆时针度量，缺省值是 $2\pi$。起始角和终止角的含义见图9.11。

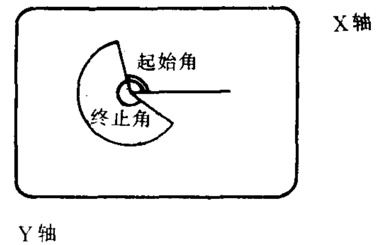



图 9.11

纵横比用来画出不同形状椭圆。如果要画圆，要选择适当的纵横比。由于屏幕上相邻的纵向的二点间距离大于横向二点间的距离，因此纵横比不应指定为 1。如果不指定此参数，系统会自动按“画圆”加以指定：中分辨显示时指定为 5/6，高分辨显示时指定为 5/12。如果自己指定此参数，请注意 $y$ 方向上的半径等于 $x$ 方向上的半径乘以纵横比（但应同时考虑屏幕上纵向的一个单位与横向的一个单位不等长）。

另外要说明的是当语句中的起始角和终止角取负值时，圆弧将通过半径与圆心相连，具体见例9.12。

**【例 9.12】**用不同颜色画一组同心圆和不同纵横比的圆弧。

```
5   REM Using CIRCLE Statement
10  SCREEN 1, 0 : COLOR 0, 1 : CLS
20  FOR R=1 TO SO STEP 5
30    CIRCLE (160, 100), R, R MOD 4
40  NEXT R
50  PI=3.14159
60  CIRCLE (80, 80), 30, 2, 0, PI/2, 1
70  CIRCLE (80, 130), 30, 2, 0, PI/2, 0.2
80  CIRCLE (240, 80), 30, 2, -PI/2, -.000001, 1
90  CIRCLE (240, 130), 30, 2, -PI/2,-.000001,0.2
100  IF INKEY$ = "" THEN 100
120  SCREEN 0 : WIDTH 80
130  END
```

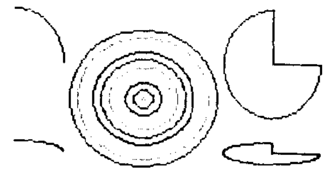


图 9.12

程序中，30 行 CIRCLE 语句中的半径是可变的，颜色通过求余运算分别得1，2，3，0，1，2，3，0，1，2 等，即彩色码在 0～3 之间变化，这样通过循环语句就可得到半径不同，4 种不同颜色的 10 个同心圆，但我们看到的只有 8 个，这是因为彩色码为 0 时，圆弧的颜色与底色相同，看不出来。另外，还有 4 个圆弧，半径都是 30（即 $x$ 向的半径），左边两个圆弧的起始角都是零，终止角都是 $\pi/2$，但纵横比不一样，上面那个圆弧是 1，由于屏幕 $y$ 轴相邻点的距离大于 $x$ 轴相邻点的距离，所以是个 $y$ 轴拉长了的椭圆弧；而下面那个的纵横比是 0.2，所以是个扁的椭圆弧。

下面分析右边的两个圆弧。要强调的是要在圆弧和圆心之间连线，则应在起始角和终止角前加负号，同时，负零应用一个负的小值来代替。图 9.12 是程序运行结果。

（1）在中分辨图形方式下，只能使用行宽是 40 个字符，即对应于“大”字符；在高分辨率图形方式下，只能使用 80 个字符的行宽，即对应于“小”字符；

（2）字符以彩色码 3 的颜色显示；

（3）图形方式下，只能使用 ASCII 码小于 128 的字符。

【例 9.13】屏幕上显示一个时钟，起始时间可由用户设定，秒针会走动并会发出响声，同时用字符显示出时、分、秒，程序如下：

```
10  REM A Clock
15  INPUT "What time is it (HH : MM : SS);", T$
20  TIME$ =T$
25  SCREEN 1, 0
30  COLOR 1, 0
35  KEY OFF : CLS
40  LOCATE 5, 12 : PRINT "12"
45  LOCATE 13, 22 : PRINT "3"
55  LOCATE 21, 12 : PRINT "6"
60  LOCATE 13, 3 : PRINT "9"
65  CIRCLE (94, 100), 80, 1,,, 7/8
70  C=6.28/60
75  J=VAL (RIGHT$ (TIME$, 2)) +45
80  X1=60 * COS (J * C) +94
85  Y1=60 * SIN (J * C) +100
90  LINE (X1, Y1) - (94, 100), 2
95  CIRCLE (94, 100), 2
100 IF T$ =TIME$ THEN 130 ELSE LOCATE 10, 9 : PRINT TIME$
110 T$ =TIME$
120 LINE (X1, Y1) - (94, 100), 0 : PRINT CHR$ (7)
130 IF INKEY$ = "" THEN 90
140 SCREEN 0 : WIDTH 80 : KEY ON
999 END
```

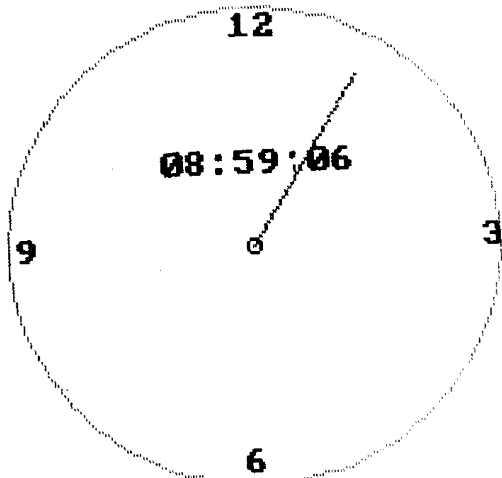



图 9.13

程序中，15～20 行语句，设定起始时间；25—30 行语句设置中分辨率彩色，屏幕底色为蓝色，0 号配色器，因此，若要用定位语句 LOCATE 在特定的地方显示字符，就应意识到一行 40 个字符。40—65 行语句画时钟；70—90 语句画秒针；95 语句画钟的中心点小圆；100 行语句显示时，分、秒；120 行语句“擦去”秒针，并发出钟响声；130 语句表示若要继续显示时钟则转去 90 行语句继续执行，如此往复，即得一个秒针会动且发出响声的时钟图形。若要结束显示，只要在键盘上按任意键就结束程序运行，回到 BASIC 的状态。

通过此例，可以学会图形方式下显示字符的方法，这在很多情况下是有用的。

程序运行结果如图9.13。

## §9.4 图 形 着 色

我们前面已经介绍过，画线语句的第三种格式，不仅可以画矩形，而且还可以在矩形内着色，这也可以说是一种着色语句。现在要介绍更一般的着色语句，它能在各种各样的复杂图形内着色，语句的一般格式是：

**PAINT (X, Y) [, <填充色> [, <边界色>]]**

其中，(X，Y) 为边界线内任意一点（称内点）的坐标，它可以是绝对坐标，也可以是相对坐标，相对坐标用前面已述的 STEP (Dx, Dy) 表示，颜色将从此点开始一直填满整个区域，直到边界为止。

填充色为区域内填充的颜色，其值范围是 0～3，缺省值在中分辨率时为 3，高分辨率时为 1。

边界色是边界线的颜色，若该参数缺省，表示对整个屏幕着色。但是当所要填充的颜色是 3，而区域边界的彩色也是 3 时，那么两者都可以缺省（如例 9.14 中的对任意多边形着色）。注意要着色的区域边界不可以有缺口，也就是说要封闭的，否则就会“漏出”，即对整个屏幕着色。

**【例 9.14】** 画一个半圆、一个三角形和一个不规则的多边形，并在其中分别涂上洋红、青色和白色。这三个图形的边界线都是由白色画成的。程序如下：

```
10  REM Using PAINT Statement
20  SCREEN 1 : CLS : COLOR 0, 1
25  '------DRAW AND PAINT A SEMICIRCLE: ------
30  CIRCLE (60, 60), 40,3,-.000001,-3.1416
40  PAINT (59, 59), 2, 3
45  ' ------ Draw and paint a triagle: -----
50  LINE (60, 80) -STEP (70, 0)
60  LINE -STEP (-15, 50)
70  LINE - (60, 80)
80  PAINT (65, 81), 1, 3
90  ' -----Draw and an irregular figurel: -----
100 LINE (200, 50) - (230, 60)
110 LINE- (210, 90)
120 LINE- (220, 120)
130 LINE- (160, 160)
140 LINE- (140, 120)
150 LINE- (200, 50)
160 PAINT (200, 51)
170 IF INKEY$ = "" THEN 170
180 SCREEN 0 : WIDTH 80
190 END
```

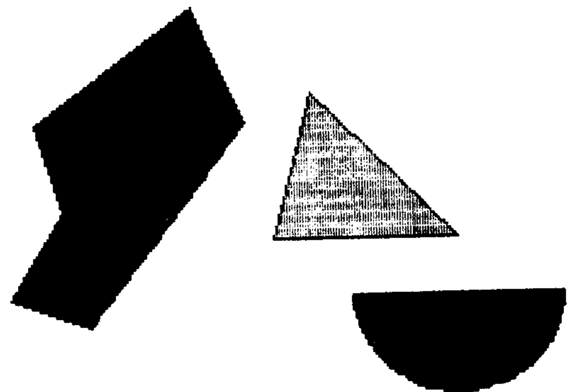


图 9.14

程序中，25～40 语句画半圆，并在半圆内涂洋红；45～80 语句画一个三角形，并涂上青色；90～160 语句画任意多边形并涂白色。程序运行结果图 9.14 所示。

**【9.15】** 利用 CIRCLE 语句和 PAINT 语句绘制扇形统计图，假定统计项目共有 9 项，每项所占的百分比分别是 13，7，17，21，6，4，11，12 和 9，画一个扇形统计图并用颜色把它们区分开来。

题目的实质是画一系列的圆弧，且每段圆弧要与圆心相连。第一个圆弧占整个圆的 13%，那么所占的弧度数应是 13/100×2π，令起始角是零度，那么终止角是 13/100×2π 弧度。同样

第 2 个圆弧占整个圆的 7%，则占弧度数是 $7/100\times2\pi$，所画圆弧的起始角应该是前面那个圆弧的终止角，所以这个圆弧的终止角应 $(13+7)/100\times2\pi$ 弧度，…一直到最后一个圆弧为止。起始角和终止角知道了，就可以画圆弧，问题是每个圆弧必须与圆心相连形成扇形，那么必须在起始角和终止角前加负号。

每个扇形画好后要着色，可用 PAINT 语句，问题的关键是要找到扇形内的一个点，本程序就是利用每个扇形的中心线离圆心 0. 8 倍的半径上，这一点总是在扇形内的，把这一点作为着色语句的“内点”开始着色。当然，可以选扇形中的其他点作为内点，只要保证它在扇形内。画圆的纵横比如果缺省，如前面已提到过的，应是一个圆，但是实际在某些显示屏上是有误差的，也就是说画出的圆不十分圆，这在例9. 12中的几个同心圆可以看出。为了得到满意的圆，可以修正纵横比来达到。其实，我们在例9. 13的画时钟时已经这样做了，我们用 7/8 作纵横比得到较好的效果。当然，不同类型的计算机，纵横比的这个修正值也可能不一样。

此程序只要修正一下数据项，就可以画别的扇形统计图。程序如下，图9. 15是运行结果(圆外的数字是另加的)。

```
100  REM A Pie Chart with colors
110  CLS : KEY OFF
150  SCREEN 1 : COLOR 0, 1
160  x=160 : Y=80
170  PI=3. 1459
180  A2=-. 000001
190  READ RAD
200  READ NCAT, TOT
210  FOR CAT=1 TO NCAT
220    READ V
230    A1=A2 : A2=A1-V/TOT * 2 * PI
240    CIRCLE (X, Y), PAD,, A1, A2, 7/8
250    PAINT (X+ (. 8 * RAD) * COS ( (-A1-A2) /2), Y- (. 8 * RAD) * SIN ( (-A1-
       A2) /2) * (7/8)), 1+CAT MOD 3, 3
260  NEXT CAT
270  LOCATE 24 : INPUT "enter carriage return to end"; X$
280  SCREEN 0 : WIDTH 80 : KEY ON
290  DATA 90
300  DATA 9, 100
310  DATA 13, 7, 17, 21, 6, 4, 11, 12, 9
320  END
```

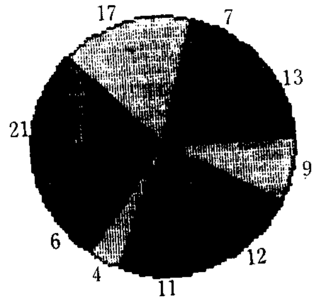


图 9. 15

此程序在黑白图形适配器的 PC 机上运行时，只要在着色语句前加一个撇号(使之变成注释语句)即可。

## 习　　题

9. 1　下面的说法对吗？若错，请纠正。

(1) 在高分辨率图形模式下，屏幕垂直方向是 640 个点，水平方向是 200 个点。

(2) 中、高分辨率图形显示模式下，屏幕的左下角这个点的坐标都是（0，199）

(3) 彩色语句 COLOR 可以使用于中、高分辨率图形显示模式，但在字符显示模式下不能使用。

(4) 中、高分辨率图形显示模式下，都允许在屏幕上有 25 行的字符显示。

(5) 假定图形模式下，现行点的坐标是（100，30），那么经过相对坐标 STEP（－33，47）以后的屏幕坐标点应是（67，77）。

9.2 在屏幕的正中部分，显示“WECOME”字样。

9.3 在图形模式下，画函数 y＝cosx，要求显示 10 个周期。

9.4 画一个扇形统计图，分 6 组，每组所占的百分比是：18、22、8、12、26、14。要求相邻两扇形用不同的颜色区分之。

9.5 设计一个竖方图，表示 1986 年到 1990 年各年生产总值的增长情况（数据自定）。

9.6 画一个红五角星。

9.7 画一面五星红旗（星的颜色为黄色）。

9.8 画两个圆，互相部分覆盖。两圆的圆心分别为（160，100），（240，100），左边的圆着黄色，右边的圆着绿色，两圆相交部分为红色。

9.9 对本章习题9.4作改进，将百分比数字“18％”、“22％”、“8％”、“12％”、“12％”、“26％”，“14％”等用字符显示在扇形区内。

# 第十章　输入输出设计

## §10.1　输入输出技术

输入和输出是程序的一个重要功能。一般说，一个程序是需要输入一些数据的，运算结果也需要输出。怎样有效地组织数据的输入输出，是程序设计人员应当仔细考虑的问题。一个好的程序应当有好的输入输出功能，它是衡量程序质量的一个重要标志。

应当考虑的有关输入输出的问题，包括：

1. 要使用户感到方便好用。即使不懂计算机语言程序设计的人也能方便地使用程序，能正确地组织数据的输入，能清晰地看出输出的结果。例如，在 INPUT 语句中用“提示”，提醒用户输入所需的数据；用“菜单”方式提供几种运算功能，用户只需简单输入一个字母或数字，程序就能执行其中一个功能（见本章§10.4）；在输入输出文字串时使用汉字以适合国内使用需要；在输出数值时采取“格式输出”，以使输出格式整齐美观；在输出时应加上必要的文字说明，如果打印结果只是一大堆数字，阅读者是很难弄清楚各个数据的含义的。

2. 要保证输入数据的可靠性。输入数据是否正确直接影响运行结果。人们常用键盘输入数据，而人工输入难免会出错，除了要求操作员尽量减少差错率以外，在程序中应当设置检查措施，以发现误操作所造成的数据错误。常用的有几种方法：

（1）对同一个数据先后输入两遍，如果二次不同，则认为可能有错，提请人们检查重输入。

```
10  INPUT "Enter your data:", A, B, C
20  INPUT "Enter your data again:", A1, B1, C1
30  IF A<>A1 OR B<>B1 OR C<>C1 THEN PRINT "ERROR": GOTO 10
```

一般说，由于操作有误而二次都出现同样的错误的机率是比较小的。当然如果操作员看错了数据而二次都错打字符，此程序是不能发现的，此方法称“复验法”，不少录入数据的系统采用此法。

（2）有些数据是有范围的，程序中应检查数据是否合法。如“日期”数据，“年”不超过1999，“月”不超过“12”，“日”不超过 31。可用以下语句检查：

```
10  INPUT "Enter Date:" M, D, Y
20  IF M>12 OR D>31 OR Y>1999 THEN PRINT "ERROR": GOTO 10
```

其它如：规定购货额不超过四位数，若超过四位数则判定数据有错；学生学号为 6 位数字，若超过 6 位则错；等等。

（3）数据加权。如果输入的数据全是数字，可以用“加权法”，即先求出该数被某整数除的余数，加在该数的后面。如输入 900818，它被 7 除的余数为 2，把 2 加在原数据之后，得

9008182。输入 9008182，由程序检查它前 6 位数被 7 除的余数是否等于 2，若不是，则肯定有错。

```
10  INPUT "Enter your data:", N$
20  IF LEN(N$) <>7 OR VAL (LEFT$ (N$, 6)) MOD 7<>
    VAL (RIGHT$ (N$, 1)) THEN PRINT "ERROR": GOTO 10
```

这种方法发现错误的可能较大。例如若把 9008182 错打成 9608182，程序检查 960818 被 7 除的余数不是 2 而是 3，判定输入有错误，除非也把检验位也打错，而且使前面 6 位数除以 7 余数恰恰等于检验位，例如 9608185。但这种错的机率很小。这种方法可靠度较高，但要事先加权，算出检验位，工作量大，所以只有在输入关键性的数据时才用这种方法。

3. 应当保证不论输入什么数据（包括错误输入的数据），都不致发生程序中断，而应由程序作相应处理。例如：

（1）如果输入的是数字，万一错输为非数字字符（例如把数字 0 错打为字母 O，把数字 1 错打为小写字母 l），如果用数值变量，如：

```
10  INPUT "Enter n:", N
```

则输入字符后便会出现“数据与变量类型不同”的错误，程序中断。为避免此情况，可以改用字符符变量，若数字长 6 位，则可用以下语句：

```
10  INPUT "Enter n:", N$
15  T=0
20  FOR I=1 TO 6
30    M=ASC (MID$ (N$, I, 1)
40    IF M<48 OR M>57 THEN PRINT "ERROR": T=1 (数字的 ASCII 码在 48 到 57 之间)
50  NEXT I
60  IF T=1 GOTO 10
```

可以将此法与复验法联合使用，既保证输入数字，又能复验，而且不致由于误输入字符而使程序中断。

（2）对不合适的输入数据要有相应处理措施。有些输入数据，输入时并无错误，而是在选择输入数据时考虑不同。例如第四章例4.5解一元二次方程式，如果输入 A≠0 时程序能给出正确结果，但若输入 A=0，数据并无错，但 X1= (−B+SQR (D)) / (2*A)，分母等于 0，溢出，发生错误。这是由于在设计程序时往往只考虑心目中正确的情况，而不考虑“意外”情况，一个好的程序应有“防御”措施，即不论操作员输入什么数据，都能作相应处理而不会使程序中断运行。应该在程序中增加对 A=0 的处理。

类似的情况是不少的，例如第四章例4.6（输入 A，B，C，求以 A，B，C 为三边的三角形面积），应当先检查 A，B，C 是否能构成三角形。如果不是三角形的三边就不应当求面积。

（3）当输入很多组同类数据时，最好不用 FOR－NEXT 循环，例如要输入全校学生的成绩数据求平均分数。如果用 FOR－NEXT 循环必须先数出要输入的数据个数，万一数错了（例如 991 人的数据，错数成 992 人），用下面语句就成问题了。

```
5   SUM=0
10  FOR I=1 TO 992: INPUT POINT: SUM=SUM+POINT: NEXT I
20  AVE=SUM/992
```

当操作员输入完 991 个数据后，程序还等待输入第 992 个数据，而实际上没有输入数据了。只

好修改程序，重输入数据，前面的输入全部白费。在这种情况下，可以用“结束标志”（如以“－1”表示输入结束），用 IF 语句或 WHILE 语句控制循环结束，这样可以不必事先去数输入数据的个数。

输入输出风格还可以有其它一些内容，读者也可以根据需要创造一些好的经验。

## §10.2 格 式 输 出

我们已介绍过在输出数据时为控制输出数据的位置而可以采用的几种方法：

1. 标准打印格式，在 PRINT 语句中用逗号分隔两个数据项。
2. 紧凑格式输出，在 PRINT 语句中用分号分隔各数据项。
3. 用输出位置函数（TAB 函数），可以控制在一行的指定位置输出一个数据项。
4. 用空格函数（SPC 函数）在输出的两个数据间插入若干个空格。如：

```
10  PRINT "BEIJING"; SPC (3); "SHANGHAI"; SPC (3); "NANJING"
```

输出为：

```
BEIJING␣␣␣SHANGHAI␣␣␣NANJING
```

只有这几种方法还不能满足用户要求，例如有时要求输出小数形式而系统却输出指数形式；有时只要求输出二位小数系统却给出 7 位小数；在输出报表时，往往要求上下行的数据按小数点对齐，但用以前介绍过的方法无法实现此要求；有时还要求在数据前加某些专用字符（如加“$”“#”，“+”字符等）。这些可以用 BASIC 提供的 PRINT USING 语句来实现“自选输出格式”（简称“格式输出”），PRINT USING 语句的一般形式为：

**PRINT USING “输出<格式字符串>”；<输出项>**

输出格式字符串（简称格式串）用来指定输出项的输出格式，它由规定的格式字符及分隔符组成。对常用使用编辑字符在下面介绍。

### 10.2.1 用 PRINT USING 语句输出数值

可以用以下一些专用的格式字符。

(1) “#”格式符

输出格式字符串中的“#”符号用来表示该位置可以由一个数字或数符如

```
10  PRINT USING "####"; A, B, C
```

若 A＝123，B＝234.5，C＝24680。输出结果为：

```
␣123-235%24680
\_A_/\_B_/\__C__/
```

可以看出，为每个输出项保留 4 列。(1) 如果数字少于“#”的个数，则前面补空格；(2) 对正数不输出“+”号，对负数输出“－”，“－”号也占据一个“#”位置；(3) 小数点后的数字按四舍五入处理；(4) 若输出项整数位数多于格式串中“#”的个数，对该数的整数部分按原样输出，但在它的前面加一个“%”，以表示“给定的位数不够”；(5) 输出项间（如 A，B，C 之间）用逗号或分号相隔，作用完全一样；(6) 可以在 PRINT USING 语句中用一个格式说明指定多个输出项的输出格式。

可以在一个格式字符串中包括两个输出格式，分别对应于两个输出项。如

10 PRINT USING "#### ␣#####"; A, B

输出结果为：

```
␣123␣␣␣-235
 A    B
```

两个输出格式（"####"与"#####"）之间如用一个或多个空格相隔，在输出时，这些空格照印。

(2) "." 格式行

用来指定小数点的位置。

10 PRINT USING "###.##"; A, B, C

若A=12.345，B=-2.3，C=1112.456，则输出为：

```
␣12.35␣-2.30%1123.46
  A      B      C
```

可以看到：若格式说明中的小数点后的"#"的个数多于输出数据的小数位数则在多余位置上补零，若它少于输出数据的小数位数则对多余数字按四舍五入处理。

(3) "**" 格式符

如果在格式说明的最左端有两个连续的"*"号（即"**"），则使数字前面的空格即以"*"允满之（这是为了防止别人在数字之前擅自增加数字），"**"的位置上可以用数字代替。

10 PRINT USING "**##.#"; A, B, C

A，B，C的值同前，输出后结果为：

```
**12.3**-2.31123.5
  A      B     C
```

(4) "$$" 格式符

在格式串中最左端如果有两个"$"号（即"$$"）则使输出的数值前面有一个"$"号。"$$"位置上可以用数字代替。

10 PRINT USING "$$##.#"; A, B, C

输出为：

```
␣$12.4␣-$2.3%$1123.5
  A      B      C
```

(5) "**$" 格式符

是前两种（"**"与"$$"）的结合，输出时在数字前加一个"$"，并且"*"代替"$"前面的空格。

10 PRINT USING "**$###.#"; A, B, C

输出为：

```
***$12.4***-$2.3*$1123.5
    A        B      C
```

(6) "+" 和 "-" 格式符

如果想对一个正数在输出时加上一个正号“+”，则在格式说明的左端放一个“+”格式符，如果在格式说明的右端有一个“+”格式符，则在输出的数值后面加上数符（正号或负号）。如果在格式说明的右端有一个“-”格式符。则在输出正的数值时该位置为空格，输出负数时该位置上有一负号。

```
10  INPUT A
20  PRINT USING "+###.#"; A
30  PRINT USING "###.#+"; A
40  PRINT USING "###.#-"; A
```

| RUN | RUN |
|---|---|
| ? 12.5↙ | ? -12.5↙ |
| ␣+12.5 | ␣-12.5 |
| ␣12.5+ | ␣12.5- |
| ␣12.5 | ␣12.5- |

如果在格式说明的左端有一个“-”号，则作为非格式字符，在该位置上原样打印一个“-”号

```
50  PRINT USING "-###.#␣";12.5, -12.5
RUN
-␣12.5␣-12.5
```

(7)“,”格式符

用来对输出数据的整数部分加分位号（从个位数起从三位数之间加一个逗号）。如果数值不超过三位数则不输出逗号，此时格式串中的逗号位置可用一个数字式替（与“#”作用相同）。格式说明中“,”的位置可以出现在小数点左边任何位置上。

```
5  A=112.4: B=3218.65: C=17621.2
10  PRINT USING "**$###,.#"; A, B, C,
20  PRINT USING "**$,###.#,"; A, B, C
RUN
```

| ***$112.4 | *$3,218.7 | $17,621.2 | ***$112.4, | *$3,218.7, | $17,621.2, |
|---|---|---|---|---|---|
| A | B | C | A | B | C |

10 语句中的格式说明内逗号在小数点之左，按上述规定输出。从格式说明中可以看到小数点左面有 7 个字符，因此每个输出数值的小数点左面也应相应地有 7 个字符（包括“*”，“$”，“,”号和数字）。10 语句的输出项 C 后面有一逗号，其作用是输出不换行，使下一个打印语句接着在同一行上输出。C 后面也可不用逗号而用分号，作用相同。20 语句的格式说明中有两个逗号，格式说明中右端的逗号作为非格式字符，在原位置按原样输出。

(8)“^^^^”格式符

在格式说明的右端如果有四个“^”符号，表示按指数形式输出。

```
10  A=728.912: B=12.48767
20  PRINT USING "##.###^^^^␣␣####.##^^^^"; A, B
RUN
```

␣7.289E+02 ␣␣␣124.88E−01

A B

请注意：格式说明中"^"前面的格式符指定了输出时指数形式数据中的数字部分的格式（而不是按标准的指数形式输出）。输出时将有效数字尽量向左端靠，正数前应留一空格，所以输出 A 的数字部分为"␣7.289"，其后的数字舍去，指数部分显然应为"E+02"，这样才使 A 的值与给定的一致。

(9)"_"格式符

下划线"_"表示其后一个字符作为文字字符原样输出。

```
10  INPUT A
30  PRINT USING "##.##_%"; A
RUN
? 37.865↙
37.87%
```

(10) 格式串中如出现非格式字符，按原样输出。

```
10  PRINT USING "THIS IS EXAMPLE _##"; 1
RUN
THIS IS EXAMPLE #1
```

注意格式串中两个"#"中的第一个由于其前有一"_"号，因此它作为文字字符原样输出，而不作为格式符号去代表一个数字位置。

## 10.2.2 用 PRINT USING 输出字符串

(1)"!"格式符

规定只显示输出项（字符串）中的第一个字符。

```
10  PRINT USING "!"; "Operating", "System"
RUN
OS
```

(2)"&"格式符

使字符串按原样输出。

```
10  PRINT USING "!"; "Chinese";
20  PRINT USING "&"; "CDOS"
RUN
CCDOS
```

(3)"\<n 个空格>\"

表示输出的字符串长为（$n+2$）。如果原来的字符串长大于 n+2，则将右端多余字符舍去，如小于 $n+2$，则输出时字符向左靠，右补空格。

```
10  PRINT USING "\␣\"; "FORMALA";
20  PRINT USING "\␣␣\" "TRANSLATION";
30  PRINT USING "\\"; "␣";
```

```
40  PRINT USING "\␣␣␣␣␣␣␣\"; "LANGUAGE"
RUN
FORTRAN␣␣LANGUAGE␣。
```

10 语句中的格式串在两个“\”之间有一个空格，因此可以输出三个字符“FOR”，同理 20 语句可以输出 4 个字符，30 语句可以输出 2 个字符（两个“\”之间无空格，即 n=0），故输出二个空格。40 语句可以输出 9 个字符，而“LANGUAGE”只有 8 个字符，故右边补一空格。

PRINT USING 语句可以这样理解：使用（英文 USING 的意思是“使用”）所指定的格式输出。PRINT USING 的规定繁多，不好记，也不必死记，用多了就自然记住了，必要时查一下手册，或上机试一下即可。

## §10.3 汉字的输入输出

到目前为止，我们用 BASIC 程序所打印出来的信息都是由英文字母、数字或一些专用符号（如+，－，*，/，?，<，>等）组成的。对于象中国这样一个非英语国家，如果打印出来的报表都是用英语或汉字拼音来表示，供人阅读，显然是很不方便的。要使计算机真正得到普及，用于各个领域（特别是事务管理领域），必须解决汉化的问题，就是说，用汉字打印出所需的信息。

怎样把一个原来不能使用汉字的计算机系统变成一个能处理汉字的计算机系统呢？这需要解决以下几个问题。

（1）确定如何用一个代码（一组字母或数字）代表一个汉字的方案。例如电报码就是一种用数字代码代表一个汉字的方法。目前国内已提出 500 多种汉字编码方案，各有优缺点。一个汉字计算机系统应当可以选用一个或几个汉字编码方案，供用户选择使用其中的一种。

（2）确定编码与二进制数之间转换的方法。我们已经知道，一个字符可以用一个二进制的代码表示（例如可以用 ASCII 代码表示一个字符），同样，需要确定如何用一组二进制代码代表一个汉字。也就是说，给定一个汉字的编码之后，如何把它转变为一组二进制的代码。

（3）需要有一个“汉字库”存放常用的汉字（实际上是汉字的代码），以备调用。汉字库可以通过两种方式实现：

①用硬件实现，装一个硬件板，即“汉字卡”，它实际上是一个包含相当数量的汉字和汉字功能的硬件板，把它插在主机的扩充槽上，就可以做到在对计算机不作任何修改的情况下，除了可以实现原有的西文功能之外，还可以处理汉字。

②用软件实现。在一张或若干张软磁盘上装入汉字和其代码，当输要处理汉字时，在用户输入某一汉字的编码之后，计算机就从磁盘上找出相应的汉字，把它调入内存。程序就可以使用它了。

国家标准 GB2312-80 规定 3755 个常用汉字为“一级汉字”，把另外 3008 个次常用汉字定为二级汉字”。

（4）为了使程序能调用汉字库中的汉字，需要对原来系统使用的软件作适当的修改，有两种不同的办法来达到此目的：

①不修改操作系统，而修改 BASIC 解释系统，使之增加处理汉字的功能。即将原来的 BA-

SIC 解释系统改成汉字 BASIC 解释系统，BASIC 中原有的全部功能都保留。

②不修改 BASIC 解释系统，而修改操作系统，这样 BASIC 通过操作系统也可以调用汉字库中的汉字。例如，IBM－PC 计算机使用的 DOS 操作系统只支持西文处理，而汉字操作 CC-DOS 则能处理中西文数据。现在很多计算机系统（例如 IBM－PC 和 APPLE Ⅱ）都有汉字操作系统，除了在 BASIC 程序中可以使用汉字外，其它语言程序也可以使用汉字。

(5) 要求有相适配的打印机（如九针打印机 FX－100，24 针打印机 M－2024 等)。

因此，并不是任何一个计算机系统都能直接使用汉字的，必须具备以上几个条件。对一个具体的计算机而言，主要应具备上述条件的 (3) 和 (4)，因为，要实现 (3) 和 (4)，必须事先确定编码方案和编码与二进制代码间的转换。如果一个计算机系统不带汉字库（硬汉字或软汉字），又没对系统软件（操作系统或解释程序）扩充汉字处理功能，则在程序中无法使用汉字。

要使程序运行时能打印出汉字，首先应该向计算机输入汉字。由于各种汉字编码方案各不相同，因此输入汉字的方法也不同。在输入汉字之前，应当先了解所用计算机选择的是哪一种汉字编码方案。

尽管编码方案有几百种，但从根本上说分成两大类，一类是“拼形法”，即根据汉字方块字的特点，将它拆成几部分，每一部分用一个字符来代表。例如 APPLE Ⅱ上使用的“仓颉编码法”，将一个“娃”字，分成“女土土”三部分，用“V”代表“女”，用“G”代表“土”，因此“VGG”就代表“娃”字。目前国内流行的“五笔字型”法也属于：拼形法”。另一类是“拼音法”，用汉字的拼音来代表一个汉字。这种方法使用方便，得到广泛的应用。

下面介绍长城 0520（IBM－PC）使用汉字的方法。

先将 CCDOS 调入计算机内存，可将 CCDOS 盘放在 A 驱动器然后用冷启动(接通电源)或热启动（同时在按“Ctrl”键＋“Alt”键＋“Del”键）即可。CCDOS 启动后，系统即进入中西文混合工作方式。这时系统允许使用 4 种汉字输入方式和一种西文输入方式。4 种汉字输入方式为：国标区位码、字形首尾码、汉语拼音码和快速编码法。西文输入方式是直接从键盘键入字符，并自动转换成 ASCII 码输入到计算机内存。不同的输入方式可以互相转换。

用以下方法选择编码方案：(同时按“Alt”键和功能键“F1”～“F6”)：

| | |
|---|---|
| ＜ALT＞＋F1 | 区位码 |
| ＜ALT＞＋F2 | 首尾码 |
| ＜ALT＞＋F3 | 拼音码 |
| ＜ALT＞＋F4 | 快速编码 |
| ＜ALT＞＋F6 | ASCII 码 |

例如，想从西文状态转为用汉字拼音输入汉字，可以同时按下键盘上的＜ALT＞键和 F3 键。如果想使用区位法输入汉字，则同时按＜ALT＞键和 F1 键。如果想从汉字输入状态转换成西文输入（即 ASCII 码输入)，可同时按＜ALT＞键和 F6 键。

下面以广泛使用的拼音输入法为例，介绍如何输入一个汉字。

一个汉字有相应的拼音，例如“初”字的拼音为“CHU”，应从键盘输入“CHU”。为了减少敲键的次数，对一些声母或韵音（如 CH，ZH……）用一个压缩代用字母来表示。下面是代用字母表：

| 拼音 | zh | ch | sh | eng | ang | ai | ing | en | ung | ao | ei | an | ong | ou |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 代用字母 | a | i | u | g | h | l | y | f | v | k | e | j | s | o |

因此，需输入“CHU”时，应输入“iu”（因“CH”用“i”代表）。先按一次“i”键，显示屏最下面一行会显示出：

拼音I：0：摘1：叉2：茬3：茶4：查5：碴6：搽7：察8：岔9：差［264］

表示以拼音“CH”开头的汉字共有274个（已显示出10个，还有264个）。你再输入字母“U”，此时显示屏显示出：

拼音IU：0：初1：出2：橱3：厨4：躇5：锄6：雏7：滁8：除9：楚［064］

表示拼音“CHU”所对应的汉字共有74个（显示出10个，还有64个，它们都是同音汉字）。你可根据需要选用其中一个。现在“初”字已在显示出的10个汉字范围之内，从显示屏上看到在这10个汉字中，“初”是第“0”个。按下“0”键，则表示选择“初”字，屏幕上方出现汉字“初”，这就输入完一个汉字。如果所需的汉字不在这10个汉字范围之内，可按“>”键，计算机会再显示出另外10个同音的汉字；如这10个汉字仍然不是所需的，可按一次“>”键，直到找到所需的汉字为止。在输入完一个汉字后，显示屏最下一行所显示的提示汉字消失，如你需要接着输入第二个汉字，可重复以上过程。

一个汉字的代码最多由3个英文小写字母组成。

在程序中哪些地方可以出现汉字呢？

①注释语句中。如：

10 REM 企业管理程序

②字符串中，如：INPUT“请输入姓名”；NAME

千万注意，决不要以为可以写成下面这样：

10　打印“欢迎”

所有BASIC语句的语句定义符（如PRINT，FOR，NEXT，IF，THEN，GOSUB……等）仍一律用英文表示。

我们如果想输入下面一个语句：

10　PRINT“姓　名”

在CCDOS支持下，调入BASICA，屏幕出现“OK”。此时按西文的方式输入：

10　PRINT　"

现在应该输入汉字“姓”字。接前面介绍的，同时按<ALT>键和F3键，转到“汉字输入”的状态。然后按以上介绍的办法输入“姓”字的拼音“XING”（实际上是输入“XY”，因为“ING”以“Y”代替）。从显示的一批汉字中选出“姓”字，然后按二次空格（因为一个汉字占二个西文字符宽度）。然后再用同样的办法输入“名”字。

下一步应输入双引号，应先脱离汉字输入状态转成西文输入状态（即ASCII码输入）。同时按下<ALT>和F6键，回到西文状态，敲入双引号键。至此，已输入了：

10　PRINT“姓名”

再按“回车”键，此语句就送到内存中了。用同样的方法输入其它语句。

如果在字符串中包含汉字和字母或数字字符（或其它专用字符），如“姓名（NAME）”，当然可以在输入完“名”字后按＜ALT＞和F6键回到西文状态再输入：（NAME）。但是从屏幕可以看到字母字符比汉字小一半。如果想使ASCII字符和汉字一样大（输出字符串时也一样大），可以同时按＜CTRL＞和F9键，使系统进入纯中文工作方式，然后再输入“（NAME）”，这时“（NAME）”与汉字一样大小。再按一次＜CTRL＞和F9，系统又回到中西文工作方式，可以接着输入语句中的其它字符。

除了可在程序中使用“汉字字符串”外，还可以用INPUT语句输入汉字，如：

10　INPUT A$

执行10语句时，出现“?”，可用上面的方法输入汉字。如：

? 北京↙

则将汉字字符串“北京”送给了字符串变量A$。应当说明，一个汉字在内存中占二个字节，而一个西文字符（即ASCII码表示的字符）只占一个字节。

下面简要介绍如何打印汉字：

长城0520（IBM-PC）可以打印出汉字，但必须使用相适配的打印机（如FX100，M2024等）。如果用的是M2024，则在用打字机打印汉字之前，应打入以下命：

A＞2024P ↙

此时就可打印出中西文信息。可以根据需要选择不同的字型。有16种字型（字的大小和字的长宽比不同），分别用A、B、C、D、E、F、G、H、I、J、K、L、M、N、O、P十六个字母来代表，供用户选择。此外还可以选择打印的每一行的宽度。同时按下“CTRL”键和“F10”键之后，屏幕上将显出：

打印字号（A-P）；__　　　纸宽（80—134）；

↑

光标

根据需要键入A到P之间的一个字符，然后光标自动右跳到“纸宽（80—134）；”之后，请你输入行宽。

如果用FX　100打印机，则不键入“2024P”命令，而键入“ALL9P”，即：

A＞ALL9P ↙

可以用“Ctrl”+“Prtsc”键的办法实现联机打印（与屏幕显示同步打印），也可以用“SHIFT”+“Prtsc”键实现“屏幕硬考贝”（将已显示在屏幕上的信息原样打印在纸上），但在有的兼容机上不能用这个方法。

也可以不用以上办法而在程序的打印语句中设定字形。用CHR$（27）；“I”；“＜字形代码＞”来指定字形。如

```
10  PRINT CHR$ (27); "I"; "D"; "中华人民共和国"
20  PRINT CHR$ (27); "I"; "C"; "北京"
```

10语句中的“D”和20语句中的“C”就是字形代码。它将打印出D形字（大号）“中华人民共和国”和C形字（次大号）“北京”。PRINT语句在显示屏上输出，LPRINT语句在打印机输出。

有关汉字输入输出，只作简单介绍，读者在使用时如遇到问题，可查阅有关手册。

# §10.4 “菜单”技术

一个较大的应用程序往往包含若干个相对独立的模块，每一个模块实现一个功能。例如“学生成绩处理程序”可以包括：(1) 输入数据；(2) 修改数据；(3) 求各班平均成绩；(4) 求各科平均成绩；(5) 找出不及格的学生名单。用户在使用这个程序时有时只需要用到其中一个功能（例如只需求各科平均成绩），这时用户可以指定程序执行某一模块。

为了方便用户，程序往往设计成在屏幕上列出可供用户选择的功能，让用户选择并输入有关信息，程序就去执行相应的模块。例如，在屏幕上显示出如图10.1所示的信息。

```
请你选择：
1. 输入或修改数据
2. 统计各班成绩
3. 统计各科成绩
4. 打印不及格学生名单
5. 退出
请输入选择的项目号：__
```

图 10.1

```
请你选择：
1. 打印全部不及格名单
2. 打印英文不及格名单
3. 打印物理不及格名单
4. 打印数学不及格名单
5. 退出
请输入选择的号码：__
```

图 10.2

这种方式很象餐馆中的“菜单”(顾客可根据菜单点菜)，用户只需输入 1—5 之间的一个数字，程序就去执行该数字所代表的功能模块。这种方式是采取人-机交互方式，有充分的提示信息，使用方便灵活，用户可不了解程序的内部结构，只需根据“菜单”的提示输入一些简单的数据即可调用有关功能。现在许多应用程序都广泛应用这种“菜单”技术。

可以有多级菜单。例如在用户敲入“4”要打印不及格学生名单。此时程序又在屏幕上显示如图10.2所示的二级菜单。

用户如输入“2”，则程序打印英文课不及格的名单。

实现菜单方式的基础是程序的模块化和层次化。程序设计人员要善于将程序分解成若干个独立性较高的模块，一个模块中又可分为若干个小模块，这就是层次化，见图10.3.。这只是一示意性的例子。

在程序设计中要进行两方面的设计：(1) 设计“菜单”，即在屏幕适当的位置上用所需字形、所需颜色显示菜单提示文字；(2) 设计各功能模块。也就是按照模块化、层次化的要求设计程序。一般用一个子程序来实现一个功能，层次化是用子程序嵌套来实现的。

下面介绍程序的梗概：

```
10 REM  学生成绩处理程序
20 CLS: KEY OFF: LOCATE 1, 17: PRINT STRING $ (40, "*")
30 PRINT TAB (17); "*"; TAB (56); "*"
40 PRINT TAB (17); STRING $ (40, "*")
50 LOCATE 2, 25: PRINT "学␣生␣成␣绩␣处␣理␣程␣序"
60 LOCATE 4, 24: PRINT "1. 输入或修改数据"
70 PRINT TAB (24); "2. 统计各班成绩"
```

```
80  PRINT TAB (24); "3. 统计各科成绩"
90  PRINT TAB (24); "4. 输出不及格学生名单"
100  PRINT TAB (24); "5. 退出"
110  LOCATE 10, 16: PRINT "请输入选择的项目号:";
120  K$=INKEY$ : IF K$="" GOTO 120
130  K=VAL(K$):PRINT K
140  IF K<1 OR K>5 THEN GOTO 110
150  IF K=5 THEN END
160  CLS: ON K GOSUB 1000, 2000, 3000, 4000
170  GOTO 20
180  END
1000  REM "输入或修改" 子程序
      ⋮
1900  RETURN
2000  REM "统计各班成绩" 子程序
      ⋮
2900  RETURN
3000  REM "统计各科成绩" 子程序
      ⋮
3900  RETURN
4000  REM "打印不及格学生名单" 子程序
      ⋮
4900  RETURN
```

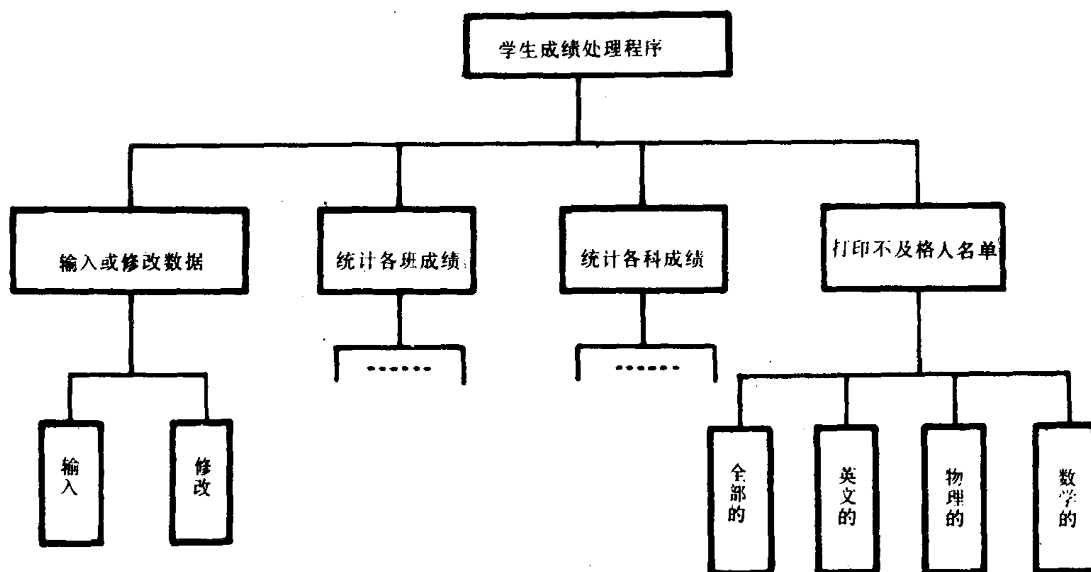



图 10.3

程序中20～110行的作用为显示"菜单"。根据用户敲入的数字（赋给变量K）决定执行哪一个子程序。执行完子程序后，返回主菜单，用户还可继续选择实现其它功能。当键入

"5"表示不需作任何工作，程序结束。请注意，在用汉字显示时，屏幕只能显示11行汉字，如用"LOCATE 12，17"是不行的。列数的计数与西文方式相同，但应考虑一个汉字占二行，空格占一行。上例也可以不用LOCATE语句定位，而只用PRINT语句和TAB函数控制输出位置。

如果有二级菜单（如图10.2所示），可在子程序（如在以4000语句为起始行的子程序）中再显示二级菜单并用ON GOSUB语句转到内嵌的子程序，执行完内嵌的子程序后返回上一层的子程序，此时应再显示二级菜单，用户可再从中选择。当不需要从二级菜单中选择。就键入"退出"项所在的项目号（如图10.2中的"5"），返回主菜单。

以上介绍的只是最基本的方法。读者可在此基础上再作进一步的加工，使之更完善美观、功能更强。例如，彩色显示；反相显示；不用键入项目号而用移动光标的方法（如把光标移到"3"处按"回车"就意味着选择"3"）；等等。

使用"菜单"技术，要综合运用有关知识（模块化、层次化、汉字输入输出、屏幕控制等），希望读者在今后编写实用的程序时能尽量使用菜单技术，以方便用户。作为程序设计人员要始终把"为用户着想"放在第一位，树立"用户第一"的思想。

## 习　　题

10.1　根据你的经验，在输入输出中最容易出现那些"事与愿违"的情况？请一一列出。用什么方法处理以避免出错：

10.2　怎样提高输入数据的可靠度？有哪些办法？比较这些办法的优缺点和适用范围。

10.3　设 $A=578.675$，$B=-8732.63$，$C=6.7234$，用下列PRINT USING语句输出A，B，C的值，得到什么结果？

(1) PRINT USING　"#####"；A，B，C

(2) PRINT USING　"###"；A，B，C，

(3) PRINT USING　"####.##"；A，B，C

(4) PRINT USING　"####.#　　###.##　　##.###"；A,B,C

(5) PRINT USING "**#####.###　　$$##.#　　**$##.##"；A，B，C

(6) PRINT USING　"+###.###　　###.##+　　##.##-"；A，B，C

(7) PRINT USING　"####,.#　　**$###,.#　　-##.##"；A,B,C

(8) PRINT USING　"#.##^^^^　　###.#^^^^　　###^^^^"；A,B,C

(9) PRINT USING　"R=##.##_%"；A，B，C

10.4　写出下面语句的输出结果：

(1)　PRINT USING "!"；"Beginner's"；"All-purpose"；"symbotic"；"Instruction"；"Code"

(2)　PRINT USING　"\␣␣␣\"；"China daily"；"BASIC Language"；

(3)　PRINT USING　"&␣"；"BASIC"；"Program"

(4)　PRINT USING　"This is a &␣"; "BASIC"; "Program!"

10.5　要在计算机系统上使用汉字需要具备哪些条件（软、硬件条件）？请考察你所用的系统具备了哪些条件？能否使用汉字。

10.6　在 BASIC 程序中哪些地方可以使用汉字？哪些地方不允许使用汉字。

10.7　你所用的计算机系统能够使用哪几种汉字编码方案。你认为各有什么优缺点？你认为哪种比较方便易学。

10.8　在屏幕中央用大字显示"中华人民共和国"

10.9　用四种大小不同的字形在屏幕同时显示"BASIC 程序设计"字样。

10.10　完成本章§10.4介绍的程序，并运行之（写成完整的程序）

10.11　在上题的基础上，变化菜单的形式：

(1) 将图10.1的第一行和最后一行用反相显示（即背景和前景的颜色互换）。

(2) 将 (1) 的要求改为闪烁形式。

(3) 不用键入数字的方法而采用移动光标的方法，将光标移到某一项目号处按回车键即表示选择该项。

10.12　请自己设计一个程序，用菜单技术。程序内容是"小学生数学游戏"，包括 (1) 程序给出两个二位随机整数，让小学生回答这两个数的和。(2) 程序给出两个两位整数，要求相减（注意应该大数减小数）。(3) 一个一位整数和一个二位整数相乘。(4) 一个二位整数除以一个一位整数，求其商的整数部分。以上四个部分都各出 10 题，每题回答一次，对一题得 10 分，最后给出成绩。从菜单中可任选一种游戏，可任选多次（多种），直至不想玩为止。

# 第十一章　文　　件

## §11.1　文件的基本概念

到目前为止，我们在运行程序时所用的输入输出设备主要是终端（显示器和打印机）。程序和数据在运行时临时输入，用完后从内存消失。下次再需要运行时再重新输入程序和数据。这是很不方便的。现在几乎所有的计算机系统可以使用磁盘（硬盘或软盘），可以将程序和数据存贮在磁盘上，需要用时再从磁盘上将程序调入计算机内存，运行时所需数据亦从磁盘上读取。这样能使程序与数据长期保存备用，又能做到数据共享（存放在磁盘上的数据可以为多个程序使用）。

一般所说的“文件”指的是“外部文件”，它是存放在外部介质上数据的集合。一批数据是以“文件”的形式存放在外部介质上的，对存贮在外部介质上的数据的存取都是以文件为对象进行的。一个源程序存放在磁盘上，就是一个“源程序文件”，每一个文件都有一个文件名以资识别。打印在纸上的一个程序，可以称为一个“打印文件”。

### 11.1.1　文件的分类

可以从不同的角度对文件分类：

1. 从文件的内容来区分，可以分为程序文件和数据文件两大类。前者存贮的是程序（可以是源程度或目标程序），后者存贮的是程序运行时所用到的输入或输出的数据。

2. 从存贮信息的形式来区别，可以分为 ASCII 文件和二进制文件。前者是以 ASCII 代码（字符代码）形式存放的，后者是以机内存贮数据的形式（二进制存贮形式）存贮的。

3. 从文件的组织形式来区分，数据文件可分为顺序文件和随机文件（见本章§11.3和§11.4）。

4. 按存贮介质来区分，可以分为：磁盘文件、磁带文件、打印文件、卡片文件等。

### 11.1.2　文件与记录

一个文件是由若干个记录（Record）组成的。也就是说，一组数据构成一个记录，若干个记录构成一个文件。因此，文件也可认为是记录的集合，记录是数据的集合。从文件中读出数据是以记录为最小单位的，也就是说，一次读取一个记录中所包含的全部数据（而不能只读记录中的一部分）。记录的长度以字节表示。MS BASIC 允许一个记录的长度为 1～32767 个字节。

### 11.1.3 文件名

为了标识一个文件，每个文件都应该有它的文件名。同一介质上不能有两个相同的文件名。MBASIC 规定文件名的形式为：

[<**盘符**>:] <**文件名**> [. <**扩展名**>]

盘符——即磁盘所在的驱动器号，通常规定 A 代表第一磁盘驱动器，B 代表第二台磁盘驱动器。盘符后一定要加冒号。当盘符和冒号缺省时，隐含指当前正在工作的“当前驱动器”。

文件名——实际上是文件名的主干，一般由 1～8 个字符组成，多数情况下，由字母和数字组成，如 BU08，A—1，PROGRAM，99TH 等都是合法的文件名主干。当然，也可以应用一些专用字符，如＄＄＄＄，aaa 等，也是合法的，但要注意有些专用字符是不可用的，如空格和逗号等，这里我们提倡用字母和数字，且长度是 1～8 个。

扩展名——其长度不超过三个字符，用来表示文件的属性，它必须用“.”号与文件名分隔。如

. BAS　表示 BASIC 源程序
. FOR　表示 RORTRAN 源程序
. COM　表示可执行的二进制代码文件
. SYS　表示系统文件
. LIB　表示库文件
. DAT　表示数据文件

扩展名所用字符的规定与文件名主干一样。但它也可以省略不写。

### 11.1.4 文件号

要对磁盘上的一个数据文件进行读写操作，必须首先开辟一个内存缓冲区，以建立必要的通道。而文件号就是规定了对这个文件进行操作的一个通道，所以也称通道号。一个文件号可以是一个数字、变量或表达式，它的取值从 1～$n$。这里 $n$ 是允许文件打开的最多数目。IBM - PC 上使用的 BASICA，在通常的情况下允许同时打开文件个数是 3 个。如果要求同时打开文件个数超过 3 个，那么，可以在启动 BASICA 时，用下列命令来实现：

BASICA/F：n↙

n 可取值 4～15，也就是说最多可同时打开 15 个文件。

例如：　BASICA/F：8↙

使用此命令启动 BASICA 以后，在以后的文件操作中，可以同时打开 8 个文件，也就是 8 个通道，可以说允许同时建立 8 个内存缓冲区。

## §11.2 源程序文件

一个源程序如果不需要存贮在外部介质上，可以不必为它定文件名。对于那些准备存贮

在外部介质上的源程序，最好在编写程序或向计算机敲入源程序时，在程序的开头用注释语句说明它的文件名以及用途，这对文件的存取和增加程序的可读性是有好处的。

如何将一个源程序以文件的形式存入磁盘；用什么方式存入；要用时又如何从盘中取出送入计算机内存；在不需时又如何从盘中删除等，对这些操作是应该熟悉的。下面我们介绍源程序文件常用的几条命令，用它们来完成保存、装入、运行、合并、删除、改名及列文件目录等操作。

### 11.2.1 SAVE 命令

把内存中一个 BASIC 源程序作为文件写入磁盘，它的一般格式是：

**SAVE "＜文件名＞"[，A]**

或 **SAVE "＜文件名＞"[，P]**

其中，文件名上节已经叙述。若文件名中的盘符省略，则文件写入当前驱动器号内的软盘或硬盘中，若扩展名省略，则系统会自动加上·BAS 作为源程序的扩展名。

参数 A 表示以 ASCII 码的形式把程序文件写入磁盘，参数 A 省略时则以二进制机器码保存文件，它可以占据较小的贮空间。不过，当要用到 MERGE 命令（下面要讲的）合并文件时，文件必须用 ASCII 码的形式存盘。

参数 P 表示以密码二进制的形式保存文件。这是一个保护性的选择项，在人们将程序再次调入内存使用它时，只能运行它。试图发出 LIST 命令或 EDIT 命令都会失败，并发出报错信息"illegal function call"（非法函数调用）。没有任何办法使这样一个程序文件"失去保护"，要么删除它。命令：

SAVE "ABC"

表示把内存中的 BASIC 程序以 ABC 作为文件名写入到当前使用的驱动器中的盘上，扩展名省略，但系统会自动加扩展名·BAS。

SAVE "B：PROG"，A

表示把内存中的程序以 ASCII 码的形式写入 B 驱动器中的软盘上，文件名为 PROG，扩展名省略。

SAVE "A：MYFILE. BAS"，P

表示把内存中的程序以 MYFILE. BAS 为名写入 A 驱动器中的软盘上，程序内容保密，但用户可以运行它。

### 11.2.2 LOAD 命令

把程序从指定的驱动器磁盘中读入（装入）内存。它的一般格式是：

**LOAD "＜文件名＞"[，R]**

带有参数时，表示程序读入内存后，立即运行程序。

例如： LOAD "ABC"

表示把当前驱动器磁盘中的文件 ABC 读入内存

表示把 B 驱动器磁盘上的文件 PROG 装入内存，并立即运行它。

LOAD “A：MYFILE. BAS”

表示把 A 驱动器磁盘上的文件 MYFILE. BAS 装入内存。

### 11.2.3 KILL 命令

删除磁盘上的一个文件。它的一般格式是：

**KILL “<文件名>”**

当然，用此命令要谨慎，不要删除有用的文件。

### 11.2.4 NAME 命令

把旧文件名改为新文件名。它的一般格式是：

**NAME “<旧文件名>” AS “<新文件名>”**

例如 NAME “YOURFILE. BAS” AS “MYFILE. BAS”

把当前盘中文件 YOURFILE. BAS 改名为 MYFILE. BAS。当然，下面这样的使用是错误的：

NAME “A：BUOL. BAS” AS “B：AIOL. BAS”

因为原来在 A 驱动器的文件，用 NAME 命令是不可能移到 B 驱动器中去的。所以，改名只能在本驱动器的磁盘上进行。注意改名时新旧文件名都必须写上扩展名。

### 11.2.5 FILES 命令

显示磁盘中的文件目录。一般格式是：

**FILES** [“**<文件名>**”]

若没有文件名，则显示盘中所有的文件目录；若有文件名，则只显示文件名指明的文件目录。

文件名还可用“*”简化，如命令：

FILES “A：*. BAS”是显示 A 驱动器磁盘上所有以“BAS”为扩展名的文件目录。

### 11.2.6 MERGE 命令

把磁盘上的一个 ASCII 码程序文件合并到内存中的当前程序中去。一般格式是：

**MERGE “<文件名>”**

例如： MERGE “B：MYFILE”

执行此命令后，就把 B 驱动器磁盘中的程序文件 MYFILE 与内存中的程序合并。如果磁盘文件程序的行号与内存中程序的行号相同，则磁盘程序的行将替代内存中程序的相应行。如果合并的程序不是以 ASCII 码形式保存的文件，那么，发出此命令后，就会出现如下报错信息：

"Bad file mode"

程序合并失败。正确应用这条命令，对于需要多次输入的大型程序是很有用的，下面举一个简单的例子来模拟大型程序，先假定在某天有一段程序已经输入计算机的内存，但由于某种原因来不及把程序全部输完，此时最好的办法是把已经输入计算机内存的程序段以文件的形式贮存到磁盘中。当以后需要继续这个工作时，可以将还未输入的程序段输入内存，然后用该命令实现合并。下面几段程序和几个命令表示了模拟上述情况的全过程。其中 new 命令是用来模拟已相隔一段时间后重新开机时内存是空的。

```
10  PRINT "Our pogram is very long. "
20  PRINT "It should separated several"
30  PRINT "parts for input computer.  1989.  4.  28"
SAVE "xxx", a
Ok
NEW
Ok
40  PRINT "Today I must finish"
50  PRINT "the program.  1989.  8.  26"
MERGE "xxx"
Ok
LIST
10  PRINT "Our program is very long. "
20  PRINT "It should separated several"
30  PRINT "parts for input computer.  1989.  4.  28"
40  PRINT "Today I must finish"
50  PRINT "the program.  1989.  8.  26"
Ok
```

### 11.2.7 RUN 命令

运行一个程序或把磁盘中的一个程序装入内存并运行它。一般格式是：

**RUN** [<**行号**>]

或 **RUN** "<**文件名**>" [, **R**]

第一种形式大家是熟悉的，命令不带行号，表示从程序的最小行号开始运行程序，带行号表示程序从指定的一行开始运行。第二种形式与前面说明的命令 LOAD "文件名" [, R] 等效。

## §11.3 顺序文件

顺序文件可以通过程序输出数据来建立。由程序输出数据以建立顺序文件时，记录总是从文件的开始处起，一个接着一个顺序地写入磁盘。当需要从文件上读出某个记录时，也必须从第一个记录开始逐个读该记录前的所有记录以后，才能读到所需要的那个记录。也就是说，顺序文件中的记录，其建立顺序、排列顺序、读出顺序三者一致，或者说，逻辑顺序与

物理顺序一致。此外，BASIS 还提供了在已建立的顺序文件后面读写一些记录的功能。读、写或续写一个顺序文件时，必须通过文件打开语句中的不同的读写方式来实现。

### 11.3.1 顺序文件的打开和关闭

顺序文件的打开语句有二种格式：

**格式（1）：**

**OPEN“＜文件名＞”FOR＜读写方式 I＞AS［#］＜文件号＞**

文件名和文件号上一节已经说明，而读写方式 I 可以取下列三种情况之一：

OUTPUT　说明顺序输出方式（写顺序文件）

INPUT　说明顺序输入方式（读顺序文件）

APPEND　说明顺序输出方式，与 OUTPUT 不同的是文件的记录指针定位在数据文件的结束处，是续写文件的记录。

格式中的“#”可以写，也可以不写。

**格式（2）：**

**OPEN“＜读写方式 II＞”，＜#＞＜文件号＞，“＜文件名＞”**

其中，读方式Ⅱ取下列二种情况：

O——说明顺序输出方式

I——说明顺序输入方式

例如：

20　OPEN“A1. DAT”FOR OUTPUT AS#1

这是用格式（1）写的文件打开语句，说明是一个新的顺序文件，文件名是 A1. DAT，准备从头开始顺序写入，文件号（通道号）是 1。当然也可以用格式（2）来写：

20　OPEN“O”，#1，“A1. DAT”

以上两个 OPEN 语句是完全等效的。应该强调指出，应用这二个语句的目的是为了建立一个新顺序文件，也就是说，磁盘里还不曾有过此文件名。假定磁盘中有此文件名，执行 OPEN 语句后，将完全破坏原有的数值，并等待着新的顺序数据的输入。

假定要在旧文件的后面增加一些数据记录，那么，只能用格式（1），此时的读写方式 I 应用 APPEND 代替。如：

40　OPEN“B. DAT”FOR APPEND AS　#　3

文件使用结束以后，一般都需要关闭，以释放所占的通道号，关闭语句的一般格式是：

**CLOSE**［［#］**＜文件号＞**［，［#］**＜文件号＞**……］

例如：

100　CLOSE #1，#2

表示关闭与通道号 1 和 2 相联系的文件，以上语句也可以写成：

100　CLOSE 1，2

当文件号缺省时，即只写：

100　CLOSE

时，则表示关闭所有的打开文件。

在执行 END，NEW，SYSTEM 或不带开关 R 的 RUN 命令以后，系统将自动关闭所有打开的文件。

### 11.3.2 顺序文件的输出（写顺序文件）

向顺序文件写数据，MS BASIC 提供了三个语句，它们是：PRINT # 语句，PRINT #　USING 语句和 WRITE # 语句。

（1）PRINT # 语句和 PRINT #　USING 语句

PRINT # 语句的格式是：

**PRINT # <文件号>，<输出项表>**

PRINT # 语句向磁盘文件所写数据的格式，和 PRINT 语句向显示器输出的格式相似。输出表项中项与项之间用逗号或分号隔开。用分号或逗号的区别与以前介绍的相同。如：

```
20  PRINT#1, A; B; C; D
```

A，B，C，D 的值以紧凑格式存放在磁盘文件 # 1 中。

如果用逗号作为分隔符，那么，各个输出数据之间的那些额外的空格也将被送入文件，这将浪费磁盘存贮空间。

在输出数值时，如果用分号分隔各输出项，由于数值之间有一个空格存在，因此当以后再将该输出数据作为输入数据被程序读入时，数据之间的空间将起到分隔数据的作用。而输出字符串时如果用分号分隔各输出项，输出的字符串之间是不留空格的，例如：

```
100  PRINT#1, "UNIVERSITY"; "OF"; "TECHNOLOGY"
```

写在磁盘文件中的信息为：UNIVERSITYOFTECHNOLOGY。以后读取时，无法区分出原来的三个数据。正确处理这个问题的方法是在输出表项中插入分隔符，即

```
100  PRINT#1, "UNIVERSITY"; ","; "OF"; ","  ";"  "TECHNOLOGY"
```

这样写到文件中去就成为：UNIVERSITY，OF，TECHNOLOGY。以后从磁盘中读回数据时，由于它们之间有一逗号，便能分别读入了。

PRINT #　USING 语句格式是：

**PRINT # <文件号>，USING "<输出格式字符串>"；<输出项表>**

它的使用与 **PRINT USING** 语句相似，只不过前者把数据写入磁盘，而后者把数据输出在显示屏或打印机上。

（2）**WRITE #** 语句

它的格式是：

**WRITE # <文件号>，<输出项表>**

用它来写文件时，能自动地在各数据项之间插入逗号，并给字符串加引号，且不在正数前面设置空格。例如：

```
100  WRITE#1, "UNIVERSITY"; "OF"; "TECHNOLOGY"
```

写入磁盘的内容是："UNIVERSITY"，"OF"，"TECHNOLOGY"。这样就无须使用专门的分隔符了。

**【例 11.1】** 建立一个 1～3 月份的收入和支出文件，程序如下：

```
10  'creat a I-E file（以撇号'开头的语句是注释语句，与 REM 等价）
```

```
20 OPEN "inex. dat" FOR OUTPUT AS #1
30 READ M$, INCOME, EXPENSES
40 IF M$ = "end" THEN 70
50 WRITE #1, M$, INCOME, EXPENSES
60 GOTO 30
70 ' close the file
80 CLOSE #1
90 END
100 DATA jan, 1000, 800
110 DATA feb, 1200, 990
120 DATA mar, 800, 950
130 DATA end,,
```

说明：20语句把数据文件INEX. DAT以OUTPUT的形式打开，建立一个新文件。若该文件名在磁盘中已存在，则删去原来的文件，并重新建立一个新文件。50语句把已读入的一行数据写入文件，直到读数结束，程序转去70行，关闭文件并结束程序运行。新建立的数据文件打印如下：

```
A>type inex.dat
"jan", 1000, 800
"feb", 1200, 990
"mar", 800, 950
```

若把50语句改为"WRITE#1，M$；INCOME；EXPENSES"，则输出结果与上面相同。若50语句改为PRINT#1，M$；INCOME；EXPENSES则运行结果是：

```
A>type inex. dat
jan 1000 800
feb 1200 990
mar 800 950
```

区别在于字符串没有引号，数字前留有符号位，数字后留一个空格。再把PRINT#语句后面的输出项用逗号分隔，程序运行结果读者可以自行分析。

### 11.3.3 顺序文件的输入（读顺序文件）

从顺序文件输入，就是从已建立的文件中读数据，这就要用到文件的读语句，常用的读语句是INPUT#语句以及还有LINE INPUT#语句。

（1）INPUT#语句

格式是：

**INPUT#<文件号>，<变量>［，<变量>］……**

其功能是从一个顺序文件读数据，并把这些数据赋值给变量。其中，文件号必须是OPEN语句中的文件号，即只能对已打开的文件才能实现读操作。变量可以是数值变量或字符型变量，也可以是数组元素。

**【例 11.2】** 读例11.1程序建立的收支数据文件，程序如下：

```
10  ' read I-E file
20  OPEN "inex. dat" FOR INPUT AS #1
30  IF EOF (1) THEN 70
40  INPUT #1, M$, INCOME, EXPENSES
50  PRINT M$, INCDME, EXPENSES
60  GOTO 30
70  ' close file
80  CLOSE #1
90  END
Ok
RUN
jan          1000          800
feb          1200          990
mar          900           950
Ok
```

程序中，20 语句以读的方式打开文件，30 语句检查文件是否结束，其中 EOF 为标志文件结束的一个逻辑函数，它的形式是：

EOF（<文件号>）

其功能是指出文件结束的条件，其中文件号是 OPEN 语句中指定的文件号。EOF 函数用于避免"输入越界"的错误，如果在指定文件上已经到达文件结束位置，则 EOF 函数的值为－1（真）；如果没有到达结束位置，则其值为 0（假）。EOF 函数仅用于顺序文件的输入才有意义。

对于已经建立的顺序文件，允许在文件的尾部增替新的记录，这是通过打开文件时使用"增补方式"来实现的。下面举例说明。

**【例 11.3】** 增补例11.1程序建立的收支数据文件

```
10  ' APPEND the I-E file
20  OPEN "inex. dat" FOR APPEND AS #1
60  INPUT M$, INCOME, EXPENSES
70  IF M$ = "end" THEN CLOSE : END
80  WRITE #1, M$, INCOME, EXPENSES
90  GOTO 60
Ok
RUN
? apr, 1000, 800
? jun, 990, 880
? end,,
Ok
system
A>type inex. dat（回到操作命令状态）
"jan", 1000, 800
"feb", 1200, 990
"mar", 800, 950
```

"apr", 1000, 800

"jun", 990, 880

上面是程序执行的全过程，并打印了增补以后的数据文件。在增补时，我们有意漏了五月份(may) 的收支情况，这将在"顺序文件的修改"中补上。

(2) LINE INPUT#语句

格式是：

LINE INPUT#<**文件号**>，<**字符串变量**>

它的功能是从打开的顺序文件中读取一个以回车键符为结束标志的字符串，字符串的字符个数不超过 254 个。注意，读取的字符串是赋值给一个字符串变量。

**【例 11.4】** 用 LINE INPUT#语句读增补以后的收支数据文件。

程序和运行结果如下：

```
10 ' read I-E file
20 OPEN "inex. dat" FOR INPUT AS #1
30 IF EOF (1) THEN 70
40 LINE INPUT #1, MLINE$
50 PRINT MLINE$
60 GOTO 30
70 ' close file
80 CLOSE #1
90 END
Ok
RUN
"jan", 1000, 800
"feb", 1200, 990
"mar", 800, 960
"apr", 1000, 800
"jun", 990, 880
Ok
```

## 11.3.4 顺序文件的修改

顺序文件的修改（如要改写原有的记录或插入漏写的记录）通常是比较麻烦的，这是顺序文件的本身读写规则所决定的，只能从文件的开始顺序地读。而写也只有二种方式，要么从头开始写，那就是建立新文件；要么从旧文件的尾部开始写，即续写老文件。所以，要修改顺序文件，通常必须建立一个中间文件，因此，修改和插入的速度比较慢。下面我们以修改和插入收支数据文件 INEX. DAT 为例来说明修改一下顺序文件的过程。其他顺序数据文件也可以仿此程序来完成修改和插入的任务。

**【例 11.5】** 对前面已建立的收支数据文件 INEX. DAT 进行修改，要求修改二月份的收支金额，并插入五月份的收支金额。程序中文件 TEMP 用作中间文件。程序从原文件中从头开始逐一读入数据记录，不要修改的直接写入中间文件，要修改的，则重新输入新记录后再写入中间文件，并在顺序读入过程中，注意插入五月份的新记录。最后，删除老文件，把中

间文件改名为原来的数据文件名而结束。程序如下：

```
10  'Change or Insert the I-E file
20  OPEN "inex. dat" FOR INPUT AS #1
30  OPEN "temp" FOR OUTPUT AS#2
40  IF EOF (1) THEN 160
50  INPUT #1, M$, INCOME, EXPENSES
60  PRINT M$, INCOME, EXPENSES
70  INPUT "change (y/n)", R$
80  IF R$<> "y" THEN 100
90  INPUT M$, INCOME, EXPENSES
100  WRITE #2, M$, INCOME, EXPENSES
110  INPUT "insert (y/n)", R$
120  IF R$<> "y" THEN 150
130  INPUT M$, INCOME, EXPENSES
140  WRITE #2, M$, INCOME, EXPENSES
150  GOTO 40
160  CLOSE
170  KILL "inex. dat"
180  NAME "temp" AS "inex. dat"
190  END
RUN
jan              1000             800
change (y/n) n↙
insert (y/n) n↙
feb              1200             990
change (y/n) y↙
? feb, 5000, 2000
insert (y/n) n↙
mar             800             950
change (y/n) n↙
insert (y/n) n↙
apr             1000             800
change (y/n) n↙
insert (y/n) y↙
? may, 9000, 2000
jun             990             880
change (y/n) n↙
insert (y/n) n↙
Ok
A>type inex. dat
"jan", 1000, 800
"feb", 5000, 2000
"mar", 800, 950
```

"apr", 1000, 800

"may", 9000, 2000

"jun", 990, 880

用顺序文件处理，不适于处理大量数据记录。而用随机文件则方便得多。

## §11.4 随 机 文 件

随机文件也称直接存取文件或相对记录文件，其特点是组成文件的每个记录都有一个唯一的记录号。但随机文件中的记录，其逻辑顺序和物理顺序一般说是不一致的。即依次先后送入文件的记录在存贮设备上的物理位置不一定是相邻的。因此，我们可以根据需要直接存取某个指定的记录，这种文件的存取速度快，且有较大的灵活性。其次，随机文件的数据通常是以二进制代码的形式存放在磁盘上。而顺序数据文件，则是以 ASCII 码的形式存盘。所以，在许多情况下，随机数据文件比顺序数据文件所占用的磁盘空间少。这一节，我们讨论随机文件的建立、修改、检索和删除等内容。

### 11.4.1 随机文件的打开和关闭

随机文件的打开也有两种格式，且这二种格式具有相同的功能。

**格式(1)：OPEN "<文件名>" AS [#] <文件号> [LEN=<记录长度>]**

**格式(2)：OPEN "R", [#] <文件号>, "<文件名>", [记录长度]**

其中，记录长度是一个整数或整型表达式，它用来为随机文件设置记录长度，其范围是 1～32767 字节之间。通常情况下，对随机文件若省略此参数，记录长度约定为 128 个字节。这个参数对顺序文件是无意义的，而对随机文件是必要的，因为随机文件具体记录的相对位置是靠记录长度来计算的。格式（2）中的"R"表示以随机方式打开文件。

值得注意的是随机文件中实际记录长度若远少于 128 个字节时，建议在 OPEN 语句中还是具体指定一个"记录长度"为好，这样可以节约文件的存贮空间。但是，当实际记录长大于 128 个字节时，那么在起动 BASIC 时，就必须预先设置文件缓冲区的长度，使得实际记录长度少于或等于文件缓冲区的长度。文件缓冲区长度的设置是在装入 BASIC 时，选择开关参数 S 的值来实现的。如命令行：

```
BASICA/S:512
```

这样就设置了文件缓冲区长度为 512 个字节。可以设置的最大文件缓冲区长度是 32767 个字节。例如：

```
   40 OPEN "BUAI. DAT" AS 1 LEN=32
或 40 OPEN "R", 1, "BUAI. DAT", 32
```

这两个语句的作用是一样的，都是在当前盘中打开随机文件 BUAI. DAT，记录长度是 32 个字节。

文件使用结束后，也使用 CLOSE 语句关闭文件，这与顺序文件中使用的 CLOSE 语句完全一样，不再重复。

## 11.4.2 随机文件的缓冲区和 FIELD 语句

文件打开以后，BASIC 分配一个文件缓冲区，其长度为大于或等于一个记录的长度。注意，文件缓冲区的长度是在启动 BASIC 时确定的，而随机文件的记录长度是在打开语句中确定的，它们的省略值都是 128 个字节。

随机文件的数据在缓冲区中都要求以字符串形式存放。因此，在数值数据放入缓冲区前必须将它们转换为字符型数据，用后面介绍的函数将数值型数据按其在机器中的内码（二进制）形式直接转换成字符形式，即按其原来形式以一个字节当作一个字符（这样可节省存储空间），然后放到缓冲区，而后用写语句把缓冲区的内容写入磁盘。反之，从磁盘文件中读回数据时，用读语句把磁盘中的数据读入缓冲区，再把那些原来是数值的字符串转换回数值。

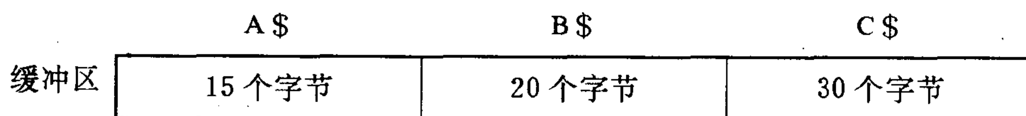
记录在缓冲区中划分为若干字段，各字段的长度一般等于记录中各数据变量的长度。即字符型变量为其字符串的长度，而数值型变量的长度规定为其机器码的长度（整型为 2 个字节，单精度型为 4 个字节，双精度型为 8 个字节）。用 FIELD 语句来规定记录中各字段的宽度。一般形式为

**FIELD[#]<文件号>,<字段宽>AS<字段名 1>[,<字段宽>AS<字段名 2>]…**

其中，字段名代表缓冲区中的字符串变量。例如：

```
10  FIELD#1, 15 AS A$, 20 AS B$, 30 AS C$
```

这个语句把文件缓冲区开始的 15 个字节分配给字符串变量 A$；把其后的 20 个字节分配给字符串变量 B$；再把其后的 30 个字节分配给字符串变量 C$。如下图所示：




FIELD 语句仅仅是给变量分配缓冲区，不能写数据，也不能读数据，所以执行了 FIELD 语句后，缓冲区还是空的。值得注意的是，在 FIELD 语句中，为变量分配的字节总数不能超过打开文件时所指定的记录长度，否则将显示错误信息：Field overflow

对于同一个文件号，可以执行多个 FIELD 语句，每执行一个 FIELD 语句，将从缓冲区的第一个字符开始重新定义字节空间的分配。但是，前后 FIELD 语句将同时发生作用，也就是说对同样的数据可以产生多种定义域。

下面我们举一个写 FIELD 语句的具体例子，假定我们要写的记录包含有关学生的下列数据项：名字、年龄、入学总分和地址四项，其中名字和地址是字符型变量，其长度假定分别是 24 个字符和 60 个字符（视最长的名字和地址而定），年龄和入学总分是数值，前者是整型，而后者是单精度实数。那么，缓冲区字段名（字符串变量）与记录的变量名及所占空间如表 11.1所示：

表11.1

| 说明 | 缓冲区变量 | 数据变量 | 字节（空间） |
| --- | --- | --- | --- |
| 学生名字 | N$ | NAME$ | 24 |
| 年龄 | A$ | AGE | 2 |
| 入学总分 | T$ | TOTAL | 4 |
| 地址 | ADD$ | ADDRESS$ | 60 |

那么，这个记录的 FIELD 语句应写成：

FIELD#1，24 AS N$，2 AS A$，4 AS T$，60 AS ADD$

记录长度是 90 个字节，这样分析后，对 OPEN 语句中的记录长度就有数了。

值得注意的是，缓冲区的字段长度分配好以后，相应数据项的长度就不应超过它的值，如上例中，若实际赋给 NAME$ 变量的名字长度大于 24 个字节时，那么在放入缓冲区时将产生截断，也就是丢失超过 24 个字符的部分。当然，假如实际数据字符串长度小于 FIELD 语句中的定义长度，则多余部分用空格填满。

### 11.4.3 随机文件的输出（写随机文件）

在把数据记录写入磁盘之前，要先用 LSET 或 RSET 语句把内存中的数据送到文件缓冲区，然后用 PUT 语句将缓冲区的内容写入磁盘。

LSET 语句和 RSET 语句的格式是：

**LSET<字符串变量>=<字符串表达式>**

**RSET<字符串变量>=<字符串表达式>**

其中，字符串变量是已经在 FIELD 语句中定义过的变量名。用 LSET 语句，使字符串变量在缓冲区的字段域中向左对齐，而 RSET 语句，使字符串变量在缓冲区的安段域中向右对齐，多余部分用空格填满。

在使用 LSET 或 RSET 语句之前，必须把数值型的数据转换成字符串，MS BASIC 提供了将数值转换成字符串的三个函数。它们是：

MKI$（<整型表达式>）

MKS$（<单精度表达式>）

MKD$（<双精度表达式>）

这三个函数分别把整型数、单精度数和双精度数分别转换成 2、4、8 个字节的二进制内码字符串。

在作了上述准备工作后，就可以用 PUT 语句将缓冲区的数据写到磁盘文件上。PUT 语句的格式是：

**PUT** [#] <**文件名**> [，<**记录号**>]

如果不指出记录号，则要写的记录就是上一个 PUT 语句所写记录的下一个记录。如：

PUT#2，28

表示将当前缓冲区的内容写入文件号为 2 的随机文件中的第 28 个记录中。接着改变缓冲区

的内容后又有语句。

PUT＃2

它没有具体指出记录号，那么，它应写入上一个 PUT 语句所写记录的下一个记录，即第 29 个记录中。

综上所述，要实现写随机文件的目的，必须执行 PUT 语句，但在执行 PUT 语句前，先要将写到磁盘上的信息置入文件的缓冲区，信息的置入可以用 LSET 或 RSET 语句来完成，而在用 LSET 或 RSET 语句之前，若记录中包含有数值型数据，又必须用 MKI＄或 MKS＄或 MKD＄函数把数值型数据转换成字符型数据。从例11. 6中可以清楚地看到这几个步骤。

**【例 11. 6】** 建立一个学生入学档案文件，每个记录包括学生的姓名、性别、年龄和入学总分。程序以 10 个学生——夏梦（Xia Meng），李伟（Li Wei），王红（Wang Hong），夏林(Xia Lin)，王时（Wang Shi），李峰（Li Feng），林芳（Lin Fang），金林（Jing Lin），王岐（Wang Qing），赵邦（Zhao Bang）为例，用 DATA 语句提供原始数据（当学生数量很多时，建议用键盘输入，以缩短源程序的长度）。程序要求每写一个记录时，在该记录的第一个字节作出写入记录的标志。程序如下：

```
10  'Creat a random file
20  OPEN "student. dat" AS #1 LEN=32
30  FIELD #1, 1 AS M$, 24 AS N$, 1 AS S$, 2 AS A$, 4 AS T$
40  READ NA$, SE$, AGE, TOTAL
50  IF NA$ = "end" THEN CLOSE : END
60  LSET M$ = "a"
70  LSET N$ =NA$
80  LSET S$ =SE$
90  LSET A$ =MKI$ (AGE)
100  LSET T$ =MKS$ (TOTAL)
110  PUT #1
115  GOTO 40
120  DATA Xia Meng, f, 19,580. 5
130  DATA Li Wei, m. 18,540. 0
140  DATA Wang Hong, m. 20. 590. 8
150  DATA Xia Lin, f, 18,520. 6
160  DATA Wang Shi, m, 19,535. 4
170  DATA Li Feng, f, 19,525. 8
180  DATA Lin Fang, m, 18,510. 6
190  DATA Jin Lin, m, 19,505. 8
200  DATA Wang Qing, f, 18,500. 5
210  DATA Zhao Bang, m, 19,520. 2
220  DATA end,,,
```

程序中，20 语句是打开一个通道号是 1 的随机数据文件 Student. dat，记录长度是 32 个字节。30 行语句用来定义文件缓冲区的字段，示意图如下：

| 1 字节 | 24 字节 | 1 字节 | 2 字节 | 4 字节 |
|---|---|---|---|---|
| M$ | N$ | S$ | A$ | T$ |
| 标志 | 姓名 | 性别 | 年龄 | 总分 |

40 语句读记录行的数据，60 语句，设置写入记录的标志为字符 a，并放入缓冲区左方第一个字节，70～80 语句把姓名和性别按向左对齐的原则放入缓冲区，90—100 语句是把原来是数值的数据项，经类型转换函数后变成字符串再按次放入缓冲区。110 语句是把缓冲区的内容写入文件。另外要说明的是写入记录标志不一定要用字符 a，可以自定别的标记。

以上所产生的随机数据文件是二进制机器码文件，是不可读的，其优点是节约磁盘空间。若要产生一个可读的随机数据文件，那么，应该把记录中的数据以 ASCII 码存盘，写入时用 STR $ 函数代替 MKI $ 和 MKS $ 函数并加大相应的字节长度。

### 11.4.4 随机文件的输入（读随机文件）

读随机文件是比较方便的。只要用 OPEN 语句打开随机文件，用 FIDLD 语句定义缓冲区的字段字符串变量，而后用读语句 GET 就可以读到文件中的任意一个你想要读的那个记录。GET 语句的格式是：

**GET ［#］<文件号>［，<记录号>］**

其中，文件号是 OPEN 语句所打开的文件号，记录号就是你要读的那个记录号，它的取值范围是 1 到 32767。如果没有指出记录号，那么，这个语句所读到的记录就是上一个 GET 语句所读的那个记录的下一个记录。

若所读的记录中有数值项，那么必须把 FIELD 语句中定义的字符串变量（二进制内码串）转换回数值，这就要用到我们在写随机文件时用过的函数 MKI $，MKS $ 和 MKD $ 的逆函数 CVI、CVS 和 CVD。它们的格式是：

CVI（<二字节的内码字符串>）

CVS（<四字节的内码字符串>）

CVD（<八字节的内码字符串>）

它们把内码字符串转换为数值。而对于以 ASCII 码存盘的可读随机数据文件，读出时应用 VAL 函数将数字形式的字符串转换成数值。

**【例 11.7】**先顺序读例11.6建立的学生档案随机文件的全部记录，而后检索文件中的任一记录。

首先提出的问题是在读随机文件 过程中，怎样来判断文件已经结束。我们在讲顺序文件时，是用 EOF 函数来判断的，但是，随机文件不可以使用 EOF 函数来判别文件的结束，而只能用测量文件长度函数 LOF 来推算。它的格式是：

**LOF（<文件号>）**

其中文件号是被打开的随机文件的文件号，LOF 函数可以得到指定随机文件 总长度（总的字节数），所以文件记录的总数是：

LOF（）/记录长度

如果按记录号逐个读入记录，则读完全部的记录后应结束。

程序如下：

```
10  ' read and retrieve the student file
20  OPEN "student. dat" AS #1 LEN=32
30  FIELD #1, 1 AS M$, 24 AS N$, 1 AS S$, 2 AS A$, 4 AS T$
40  PRINT TAB (2) "m"; TAB (5) "name"; TAB (29); "sex"; TAB (34) "age"; TAB (38) "total";
    TAB (46) "rn"
60  FOR I=1 TO LOF (1) /32
90     GET #1, I
100    MM$=M$; NA$=N$; SE$=S$
110    AGE=CVI (A$); TOTAL=CVS (T$)
120    PRINT TAB (2) MM$; TAB (5) NA$; TAB (29) SE$; TAB (33) AGE; TAB (37) TOTAL;
       TAB (45); I
130  NEXT  I
140  INPUT "Which record======>"; R1%
150  WHILE R1%>0 AND R1%<=LOF (1) /32
160    PRINT TAB (2)  "m"; TAB (5)  "name"; TAB (29)  "sex"; TAB (34)  "age"; TAB (38)
       "total"; TAB (46) "rn"
170    GET #1, R1%
180    MM$=M$; NA$=N$; SE$=S$
185    AGE=CVI (S$); TOTAL=CVS (T$)
190    PRINT TAB (2) MM$; TAB (5) NA$; TAB (29) SE$; TAB (33) AGE; TAB (37) TOTAL;
       TAB (45) R1%
200    GOTO 140
205  WEND
210  CLOSE
220  END
 RUN
 m name                                   sex      age     total     rn
 a  Xia Meng                              f        19      580. 5    1
 a  Li Wei                                m        18      540       2
 a  Wang Hong                             m        20      590. 8    3
 a  Xia Li                                f        18      520. 6    4
 a  Wang Shi                              m        19      535. 4    5
 a  Li Feng                               f        19      525. 8    6
 a  Lin Fang                              m        18      510. 6    7
 a  Jin Lin                               m        19      505. 8    8
 a  Wang Qing                             f        18      500. 5    9
 a  Zhao Bang                             m        19      520. 2    10
which record======>? 9
 n  name                                  sex      age     total     rn
 a  Wang Qing                             f        18      500. 5    9
which record======>? 5
 m  name                                  sex      age     total     rn
```

```
 a   Wang Shi                       m      19     s35. 4     s
which record=======>? 2
 m   name                           sex    age    tota       rn
 a   Li Wei                         m      18     540        2
which record=======>? -1
Ok
```

程序中，60—130 读随机文件的全部记录，其中 60 语句用到 LOF 函数来判断文件是否结束。140—205 行用来检索文件中存在的任一记录。140 语中 R1%表示 R1 是整型变量，%是整型类型符号，见第十二章 § 12.2

**【例 11.8】** 检索、重写或删除随机文件中的任一记录。此例仍以例11.6中建立的学生档案为例，只要你输入记录号是大于零的，系统都给响应。但是当你给出一个记录号超出文件中写过的最大记录号，那么，系统提示文件的最大记号，并告诉你的记录号已经超过了文件的最大记录号，要求你重新输入新的记录号。当输入的记录号大于零而又小于等于文件的最大记录号时，系统显示该记录的内容，并提问此记录要重写（Update）吗？或要删除（Delete）吗？还是不改变（Nochange）。若回答是“U”，那么要求你输入重写内容，而后系统重写此记录；若回答是“D”，那么此记录就删除了，而记录删除的标志是在记录的第一个字节（也就是缓冲区的第一个字节）上赋值“D”，这样，在下次检索记录时，只要检查一下记录的第一个字节，若是“D”则就表示此为已删记录；若你回答的是“N”，那么，记录原封不动。此后，系统又要求你检索新的记录，如此循环，可以检索、重写或删除文件内的任一记录。这里所说的重写，有两层意思，一是原有记录的改写（Update），二是已删记录的重写（Rewrite）。本程序同时有这两种功能。要退出循环，只要输入一个小于零的记录号，系统就退出运行。

本程序较全面地反映了随机文件存取的灵活性，基本上包含了我们在随机文件这部分所学的内容。其结构化流程图如图11.1所示。

程序及运行结果如下：

```
10  ' Retrieve/Update/Delete a record
20  OPEN "student.dat" AS #1 LEN=32
30  FIELD #1, 1 AS M$, 24 AS N$, 1 AS S$, 2 AS A$, 4 AS T$
40  RECLEN%=32
50  FILELEN%=LOF (1) /RECLEN%
60  INPUT "which record=====>", R1%
70  WHILE R1%>0
80    IF R1%>FILELEN% THEN PRINT R1%; "is past the end of file————": FILELEN%: GOTO
      288
90    GET #1, R1%
100   MM$=M$ : NA$=N$ : SE$=S$
110   AGE=CVI (A$) : TOTAL=CVS (T$)
120   IF MM$ = "d" THEN PRINT "record"; R1%; "was deleted. ": INPUT "Do you want to rewrite (y/
      n)"; B$: IF B$ = "Y" OR B$ = "y" THEN GOSUB 400 ELSE 180
130   PRINT TAB (2); "m"; TAB (5); "name"; TAB (29); "sex"; TAB (35); "age"; TAB (39);
      "tatal"; TAB (47); "rn"
```

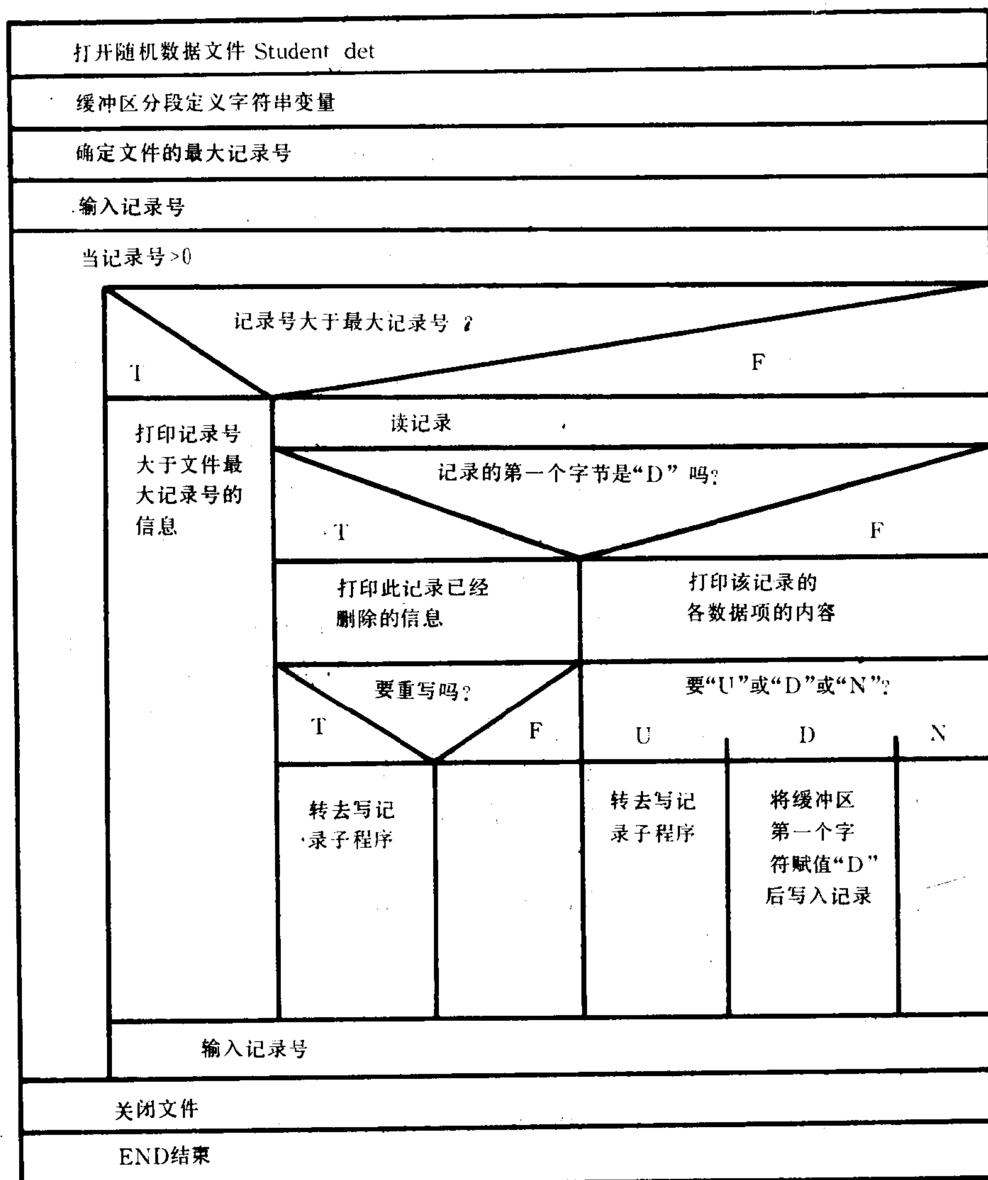



图 11.1

```
140  PRINT TAB (2); MM$; TAB (5); NA$; TAB (29); SE$; TAB (34); AGE; TAB (38); TOTAL;
     TAB (46); R1%
150  INPUT "Update (U) or delete (D) or No change (N)"; R2$
160  IF R2$ = "U" OR R2$ = "u" THEN GOSUB 400
170  IF R2$ = "D" OR R2$ = "d" THEN LSET M$ = "d": PUT #1, R1%
180  INPUT "which record====>", R1%
190  WEND
200  CLOSE
300  END
400  INPUT "New record (name, sex, age, total):"; NA$, SE$, AGE, TOTAL
```

```
410    LSET M$ = "a": LSET N$ =NA$ : LSET S$ =SE$
420    LSET A$ =MKI$ (AGE): LSET T$ =MKS$ (TOTAL)
430    PUT #1, R1%
440    RETURN
Ok
RUN
which record====>9↙
m   name                      sex age total    rn
a   Wang Qing                 f   18  500.5    9
Update (U) or delete (D) or No change (N)? n↙
which record====>7
m   name                      sex age total    rn
a Ling Fang                   m   18  510.6    7
Update (U) or delete (D) or No change (N)? d↙
which record====>3H
m   name                      sex age total    rn
a   Wang Hong                 m   20  590.8    3
Update (U) or delete (D) or No change (N)? d↙
which record====>5↙
a   name                      sex age total    rn
a   Wang Shi                  m   19  535.4    5
Update (U) or delete (D) or No change (N)? u↙
New recoed (name, sex, age, total):? Xia Wei, m, 15, 890↙
which record====>3↙
record 3 was deleted.
Do you wnat to rewrite (y/n)? y↙
New recoed (name, sex, age, total);? Wang Hong, f, 16, 986.5↙
m   name                      sex age total    rn
a   Wang Hong                 f   16  986.5    3
Update (U) or delete (D) or No change (N)? n↙
which record====>-1
Ok
```

程序开始运行以后，我们做了以下几件事：

(1) 检索第 9 号记录；

(2) 检索第 7 号记录后删除此记录；

(3) 检索第 3 号记录后删除此记录；

(4) 检索第 5 号记录后，改写它成为夏伟 (Xia Wei) 同学的记录。

(5) 检索第 3 号记录，系统显示此记录已删，我们再重写 3 号记录（仍然是王红的，但后面几项内容改变了）；

(6) 输入－1，结束运行。

以上我们对文件的操作结果，可以通过运行例11.7中的程序来检查，下面是装入并运行例11.7源程序的结果，可以看出第 7 号记录被删除了；第 3 号记录删除后又重写了新的内容；

第 5 号记录被改写了；其他记录均没有改变。

LOAD "file11-07", r

| m | name | sex | age | total | rn |
|---|---|---|---|---|---|
| a | Xia Meng | f | 19 | 580.5 | 1 |
| a | Li Wei | m | 18 | 540 | 2 |
| a | Wang Hong | f | 16 | 986.5 | 3 |
| a | Xia Lin | f | 18 | 520.6 | 4 |
| a | Xia Wei | m | 15 | 890 | 5 |
| a | Li Feng | f | 19 | 525.8 | 6 |
| d | Lin Fang | m | 18 | 510.6 | 7 |
| a | Jin Lin | m | 19 | 505.8 | 8 |
| a | Wang Qing | f | 18 | 500.5 | 9 |
| a | Zhao Bang | m | 19 | 520.2 | 10 |

which record=====>? -1

Ok

最后，我们还要介绍一个函数——获得当前记录号函数 LOC，它的一般形式是：

**LOC（<文件号>）**

LOC 函数能得到刚刚被读或写过的随机文件的记录号（当一个文件打开以后，尚未进行读或写操作时，其值为零）。它常被用于条件语句中，用来读取数据文件中的某一段记录。

## 习　　题

11.1　什么是文件？文件在程序设计中有何作用？

11.2　顺序数据文件有何特点？它是怎样进行读/写的？

11.3　随机文件有何特点？它是怎样进行读/写的？

11.4　某班 34 名学生，期末考试科目有数学、外语、计算机语言和物理四门课。试编一个程序，将这 34 名学生的姓名和学号及各科考试成绩存入一个顺序数据文件。

11.5　将上题的 34 名学生的姓名和学号及各科成绩建立一个随机数据文件。

# 第十二章　结构化程序设计方法和编程技术

## §12.1　软件工程方法和结构化程序设计

在第二章中已简要地介绍了结构化程序设计的要点和N－S结构化流程图。在前面各章中我们都是按结构化的原则来编写程序的。

结构化程序设计方法是针对前一时期的软件危机而提出的，它提倡按照工程的方法，要求程序设计人员遵循一种规范化方法进行程序设计，纠正程序设计无章可循、程序成为“个人工艺品”的状况。造成程序质量差、编程和维护效率低的主要原因是在程序中滥用GOTO语句。图2.7表示的就是一个典型的非结构化程序的流程示意。非结构化的程序可读性差，编程和调试、维护困难。结构化程序设计方法的提出是程序设计方法的一场革命。

自1965年荷兰学者Edsger W. Dijkstra提出用结构化方法进行程序设计以来，结构化程序设计的理论与技术日益完善，现在已普遍为人们接受，并成为程序设计的主流。

结构化程序设计方法主要包含：

1. 自顶向下；
2. 逐步细化；
3. 模块化；
4. 结构化编程。

这些已在前面作了简要介绍。由于前面所遇到的问题都是比较简单的，因此有些读者还未能充分领会结构化程序设计方法的实质及其实现方法。在读者学习了前面各章的基础上，在本节中我们再进一步讨论有关结构化程序设计的问题。及便今后设计较复杂的程序时有所遵循。

我们知道，程序设计的实质是以计算机为工具对问题求解。而从程序的设计目标（最后要达到的目的，即题目要求）到产生一个源程序之间要经历一段距离，研究程序设计方法的目的就是为了能够正确化地、有规律地跨越这个“距离”。对于简单的问题来说，这个距离是很短的，甚至一步就可以达到。而对于一个复杂的、大型的软件来说，就不是那么轻而易举的，甚至会在拿到问题后感到无从下手。近年来发展了一套用工程设计的方法来“生产”软件，这就是“软件工程”的方法。

我们先分析一下对一个不太复杂的问题（即它不是一个宠大的、复杂的数据处理系统）的开发过程：

(1)　对任务进行分析。分析给定的数据、需要完成的功能、对输出的要求。将它的概括地、明确地用文字整理出来，这一步称为“功能的规定”。许多人对这一步不加重视，结果当程序编写并运行之后发现与要求不符，重新返工。应该从一开始就使每一步都是明确无误

的，在保证前一步的正确性的前提下才展开后一步的工作。

(2)　根据要求，用简明的话概括写出程序最后应完成的目标，例如“求多元一次联立方程的根”、“统计全校学生成绩”、“用筛法找 2～100 间素数”等。这是一个总的任务，或者称为“宏任务”（“宏功能”）。它是高度概括的任务，它并没有具体指出应包括哪些具体的操作。这一步可以认为设计了一个“抽象的程序”，实际上是“顶层设计”。见图12.1中的“顶层”。

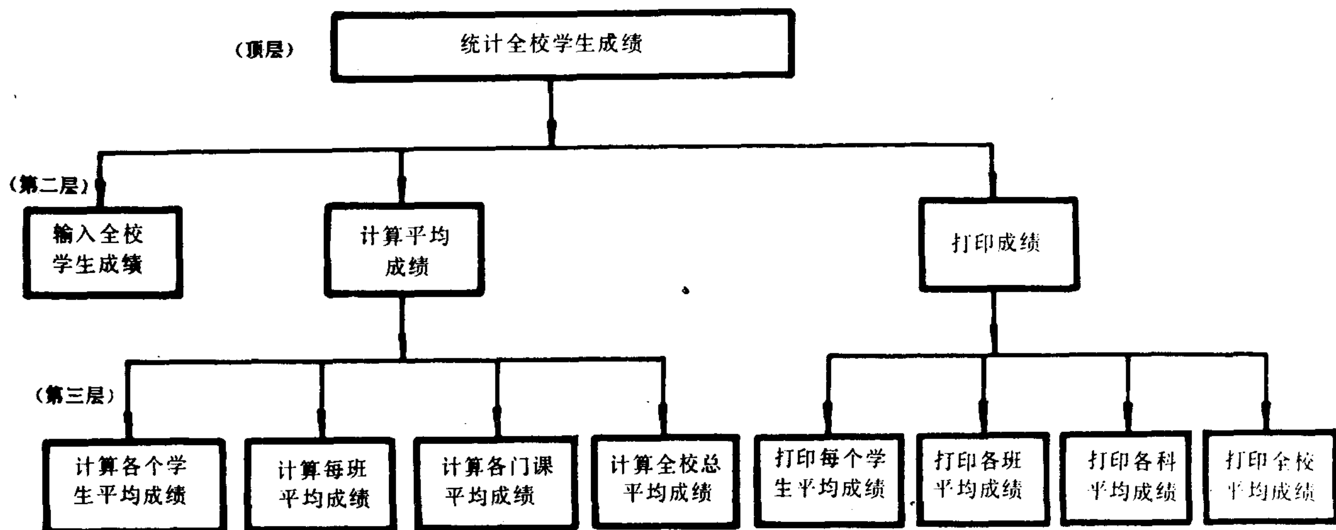



图 12. 1

(3)　将“宏任务”分解为若干“子任务”（子功能）。如图12.1的“第二层”。它比顶层具体一些了。需要说明的是，每一步由上一层向下一层的细化过程都应确保其正确性．也就是说，如果实现了第二层的子功能，就能实现顶层设计的宏功能。假设细化过程进行得不合适，例如少了中间那个“计算平均成绩”，就不能认为细化过程是正确的。

(4)　再向下分解。如图12.1中第三层。也就是将子功能再分解为更小的子功能。这个过程一直继续下去，直到每个子功能简单到不能（或不需要）再分解为止。这就是“逐步细化”。

应当说明这个“自顶向下、逐步细化”的过程不是绝对的。在细化的过程中如果发现原来的方案不理想，也可以采用局部的“自下向上”的方法予以调整。例如在图12.1中当进行到“打印成绩”这一模块的细化时，可能会发现将“打印每个学生平均成绩”等项合并到“计算各个学生平均成绩”模块一起更方便，此时可以将有关的第三层模块合并，因此需反过来修改第二层相应模块，改成“计算并打印成绩”，然后接着进行其它部分的自顶向下的逐步细化工作。也就是说，在整体的“自顶向下”过程中可以有局部的“自下向上”过程与之配合，一切依具体问题而定。

在模块划分中，应当使一个子功能单独构成一个模块，即不要把几个不同的功能放到一个模块中，也就是要使模块的“耦合程度”尽量地小，“内聚程度”尽量地大。即一个模块与外部联系（耦合）较小而与内部其它部分联系较大，这样，在需要修改程序时只需要修改某一个模块而不致于涉及所有的模块。也就是增强模块的相对独立性。模块的规模不宜过大。在程序中一般不超过 50 行（但也不是绝对的，有些模块功能不能再分解了，超过 50 行也是可

以的。）

（5） 根据模块的功能要求，分别对每一个模块写出算法。画出 N～S 流程图。

（6） 在算法设计的同时，实现数据结构的设计，即设计出各模块中使用的数据结构。

（7） 根据算法和数据结构的设计，用高级语言进行结构化编码（写出程序）。

（8） 进行程序调试，确保程序正确。

从以上过程可以看到，逐步细化是结构化程序设计中一个极其重要的组成部分，是结构化设计思想的核心，它指明了结构化设计过程所必须遵循的方法和步骤。

其实，逐步细化方法不仅适用于程序设计，而且适用于一般的工程设计，甚至社会生活中。例如，写一篇调查报告，可以采用这样的方法：先确定题目和中心思想，再将整个报告分成几个大部分以及确定各部分的题目，然后再进一步决定出每一部分应表达哪些思想要点……，直到作者认为此提纲已足够详细为止，按此就能顺利地写出报告来。

因此，可以说，逐步细化方法是关于"方法论"知识中重要的一部分。这种从抽象→具体，从宏功能→子功能，从整体→细目的分解过程，以及最后逐一实现这些细目的过程，是非常科学的、具有严密的逻辑性的。因此可以说，程序设计是人们思维活动的一面镜子。也就是说，程序员的程序构思是否有条理，是否有逻辑性，是否经过规范化训练，这些素质都要在程序中表现出来。

逐步细化方法是由"程序设计目标"走向源程序的正确途径，也就是说，用逐步细化方法才能从"提出任务"起一步一步地具体化，最后达到目的——写出源程序。

如果我们面对的不是一个简单的程序，而是一个比较大的、复杂的数据处理系统（一个系统中可能不止一个程序，而是由若干个程序组成的）。那么，进行这个系统的开发，就不仅是上面所说的程序设计过程了。从软件工程的观点，一个软件系统的生存周期应该经历以下几个阶段：

（1） 问题定义与需求分析

（2） 概要设计

（3） 详细设计

（4） 编写程序与单元测试

（5） 综合测试与确认运行

（6） 系统维护

或者说，软件生存周期由以下三个时期组成：①软件定义时期（即上面的第 1 个阶段）；②软件开发时期（上面的第 2～5 阶段）；③软件维护时期（即上面第 6 阶段）。

软件定义时期的任务是确定："要解决的问题是什么"，"有无可行的办法"，"为了解决这个问题，系统应该做什么"，即确定要开发的软件应当具有哪些功能，以及系统的运行环境等。这是进入开发时期之前必须首先解决的问题。这个时期的工作是由系统分析员和用户共同进行的，最后应提出一个完整准确的系统逻辑模型。

进入开发时期后，首先进行概要设计，这个阶段的任务是：从宏观的角度提出"怎样解决这个问题"，确定解决问题的方案，将一个大的系统分解为若干个子系统，即面向用户的"功能模块"（简称"用户功能模块"）。根据"功能模块"要求实现的功能，确定面向程序员的"程序模块"。程序模块是从程序员的角度考虑如何实现"功能模块"的要求。要为每一个程序模块写出一个"程序设计任务书"。该任务书中所叙述的程序目标、程序处理说明、算法

规定等都应当是能为程序员所理解和接受的。当程序模块的规模较大时，还需要将一个程序模块进一步划分为若干个程序子模块。

在详细设计阶段，程序员需要根据交给他的程序模块或程序子模块的“设计任务书”的要求，详细描述算法。也就是解决“怎样具体实现这个模块”。这个阶段还不是编写高级语言程序，而是详细地设计出解题的每一个步骤，可以用流程图、伪代码来表示。

在完成详细设计之后，进入编程序阶段，即用高级语言来实现详细设计阶段描述的各个操作步骤。这一工作称为“编码”。只要详细设计阶段的结果是正确的，编码阶段不应有大的问题，关键是要熟练地掌握和使用高级语言。在编写好程序之后，要分别对每一个模块进行测试，检查它们能否正确实现所规定的功能。如有错误应及时发现和改正。

然后将各个模块联结起来测试，使软件达到预定的要求。如已确认整个系统已达到预期目标，即可交付使用。

软件开发时期的工作到此结束。

在程序交付使用后，在实践中必然会发现一些问题；或者根据用户的需要，要求增加一些新的功能；或者由于运行环境的改变（例如计算机系统的更换），需要对软件作某些修改。这就是软件维护时期。这个时期是很长的，直到该软件被淘汰为止。

定义、设计与编码、测试、使用维护各个阶段所花费的工作量大体的比例为：1∶10∶50∶(50～1000)。可见，越到后面的阶段所花的工作量越大。

应当说明，以上各阶段的划分并不是很严格的，也不是一成不变的。一般意义上的程序设计指的是软件开发时期中的设计和编码，即涉及概要设计、详细设计和编码阶段。

软件工程中各个阶段一般并不是由一个人从头到底去完成的，而是分别由不同的人去完成的。一般说，问题定义与需求分析是由系统分析员完成的，概要设计是由主程序员完成的，详细设计和编写程序是由程序员完成的。测试工作是由有经验的软件人员完成的。作为程序设计人员应当了解在整个软件开发阶段中自己工作的位置与要求。

## §12.2 选择数据类型

在较早的BASIC版本中，变量（包括简单变量和数组）的类型只有数值型和字符串型两大类，数值变量不再分类，都按实型处理。考虑到程序的通用性，在本书前面各章中对数值型变量都没有再作类型说明，即把它的全部看作为实型变量。

MS BASIC（以及 Applesoff BASIC 和许多 BASIC）提供了多种数据类型，它们是：整型、实型、双精度型和字符串型。有二种方法表示数据的类型。

(1) 以变量名最后一个字符来说明数据类型。见表12.1。

表 12.1

| 变量类型 | 说明符 | 举　例 |
|---|---|---|
| 整　型 | % | A%，COUNT% |
| 实　型 | [!] | T!，AMOUNT |
| 双精度型 | # | B#，PAY# |
| 字符串型 | $ | C$，NAME$ |

例如 A 或 A! 代表实型变量，而 A%是整型变量，A#是双精度变量，A $ 是字符串变量。同样在 DIM 语句中：

```
10 DIM B(10),C! (10),D%(10),E#(10),F$(10)
```

B 和 C 是实型数组，D 为整型数组，E 为双精度型数组，F 为字符串数组。

从表12.1可知，一个变量名后面若无类型说明符，则认作实型。因此前面各章程序所用到的数值变量都是实型变量。

常量也有类型之分。MS-BASIC 规定按常量的字面形式可以区分其类型：(1) 如 1，−10，123 为整型常数，但超过−32768～32767 范围的数不作为整数，其中≤7 位的数作为实数（如 1234567 作为$0.1234567\times10^7$处理），多于 7 位的作为双精度数处理（如 123456789 作为$0.123456789\times10^9$处理）。(2) 带小数的数是单精度实数（如123.4，10.0），但超过 7 位的小数也自动作为双精度数处理。(3) 用“E”表示指数符号的数是单精度实数（如1.23E+02，相当于123.0）。 (4) 用“D”表示指数符号的数是双精度数（如1.2345678D+02表示 123.45678)。

可以在一个常量后面加上类型说明符以指定其类型，如 2%为整数，123! 为实数，3.5# 为双精度数。注意不能用 5 $ 表示 5 为字符常量。

(2) 用类型定义语句定义变量类型。

用 DEFINT 定义整型变量（DEF 是 Define（定义）的缩写，INT 是 Integer（整数）的缩写)。用 DEFSNG 定义单精度变量 (SNG 是 Single precission (单精度) 的缩写)，用 DEFDBL 定义双精度变量 (DBL 是 Double precission（双精度）的缩写)，用 DEFSTR 定义字符串变量 (STR 是 String (字符串) 的缩写)。用以上语句定义以某个字母开头的变量分别属于整型、实型、双精度型和字符串型。如：

**【例 12.1】**

```
10 DEFINT   A, B : DEFSNG  C—E : DEF DBL  F—H, DEFSTR  I, J
20 A1=10/3 : FIN=1#/3 : C2=1/3 : I= “BOY”
30 X%=10/3 : Y# =1#/3 : Z=1/3 : T$ = “BOY”
40 PRINT    A1, X%
50 PRINT    FIN, Y#
60 PRINT    C2, Z
70 PRINT    I, T$
80 END
```

分别定义了以 A 或 B 开头的所有变量为整型变量，以 C，D，E（即由 C 到 E）开头的变量为实型，以 F，G，H 开头的变量为双精度型，以 I，J 开头的变量为符串型，运行结果如下：

```
RUN
 3                      3                      (整型)
 .3333333333333333      .3333333333333333      (双精度型)
 .3333333               .3333333               (单精度型)
 BOY                    BOY                    (字符串型)
```

请注意：1#/3 得到双精度结果，而 1/3 得到单精度结果，如果将 30 行中的“Y#=1#/3”改为“Y#=1/3”，得到的是.3333333000000000，即使 Y#是双精度变量，但 1/3 只能给出 7 位有用的数字。

表12.1表示各种类型的数值型数据的有关性质。

**表** 12.1

| 类　型 | | 整　型 | 单精度实数 | 双精度数 |
|---|---|---|---|---|
| 在内存中占字节数 | | 2 | 4 | 8 |
| 表示精度 | | 精　确 | 可能有微小误差 | 可能有微小误差 |
| 有效位数 | | −32769～32767 | 输出 7 位 | 输出 16 位 |
| 表数范围 | | | $\pm(1.47\times10^{-39}\sim1.71\times10^{38})$ | $\pm(1.47\times10^{-39}\sim1.71\times10^{38})$ |
| 书写格式 | 定点 | 不含小数点 | 可含小数点 | 可含小数点 |
| | 浮点 | | 幂用“E”表示 | 幂用“D”表示 |

在程序中不同类型的数值变量可以互相赋值，赋值后按被赋值的变量的类型存储，也可以将不同类型的数值变量混合运算，运算时先将精度低的化成精度高的，然后进行同一类型的运算。

**【例 12. 2】**

```
10  I%=3.6：PRINT  I%,
20  A! =I%+4.5：PRINT  A!
30  D#=1#/3+A!  ：PRINT  D#
40  B=D#+2.34567891#  ：PRINT  B：END
RUN
 4              8.5             8.833333333333333         11.17901
```

说明：10 行中将单精度数3.6赋予整型变量 I%。由于 I%只能容纳整数，对小数部分四舍五入。20 行中“I%+4.5”，先将 I%的值转换成实型数4.0再加上4.5。30 行中，1#为双精度数，3 为单精度数，先将 3 转换为 3#，然后进行 1#/3#，得双精度结果。40 行得一个双精度数赋给单精度变量，对 7 位之后的数字按四舍五入原则截去。

也可以用类型转换函数对不同类型的数值进行互相转换。

CINT（X）　　将 X 转换成整型

CSNG（X）　　将 X 转换成单精度实型

CDBL（X）　　将 X 转换成双精度型

其中 X 可为任何类型的数值型数据，如：

```
10  PRINT     CINT（3.5），CSNG（1#/3），CDBL（3%）
```

输出：

```
 3            .3333333              3.000000000000000
```

当程序中对精度要求比较高时（7 位有效数字不够），可用双精度型。当要求数值严格准确时应该用整型。例如，循环变量如果用实型变量，有可能产生循环次数的增加或减少，如：

```
10    FOR  X=0  TO  2  STEP 0.1
20      PRINT  SIN（X）
30    NEXT  X
```

理论上应该循环（2—0）/0.1+1=21 次，但0.1的存贮是有误差的，因此每循环一次加

0.1时就会积累误差，加完 20 个0.1后可能是2.000001而不再循环了，丢失一次循环，可使用整型变量作循环变量：

```
10  FOR  I%=0  TO   20  X=I%/10：PRINT  SIN (X) ：NEXT  I
```

在程序设计时，正确选择和运用数据的类型是很重要的。双精度数的运算的精度高，但占内存单元多，运算速度慢。单精度实数次之。整型数存贮没有误差，占内存单元少，运算速度快，但数的范围小。在程序设计时应根据题目的要求、数的范围、以及所用计算机的内存容量和速度，作综合考虑。

## §12.3 程序风格

写文章有一个文章风格问题。例如有人爱用文言文，有人爱用白话文；有文章象八股文，枯燥乏味，有的文章生动活泼，引人入胜；有人爱咬文嚼字，文章涩晦难懂，有人直截了当，文章通俗易懂。对同一个内容不同人有不同的表述方式，文章风格因人而异。我们首先要求文章能使人看懂，能正确表达意思，同时希望文章风格优美，令人喜欢。同样程序也有一个风格问题。风格优美的程序结构清晰、含义清楚、输入数据易于组织、输出信息整齐美观、程序容易阅读、便于理解、使用方便。

什么是程序的风格 (stgle)，并无严格的定义。一般是指一个程序员在程序设计过程中，在程序的各个方面（例如标识符（变量）的定名、程序的书写格式、模块分解、程序语句编排、输入输出设计等）所表现出来的习惯的总和。我们从许多程序中总结出以下一些应注意的问题（其中有的在前几章中已经提及过了）。每个人都可以根据自己的经验丰富和补充它。一个风格优美的程序应该：

（1）程序模块化，一个模块实现一个功能。模块由子程序实现，模块的排列顺序一般按调用的顺序。子程序的位置一般放在程序的最后（END 语句的后面）。这样可以避免主程序的流程直接进入子程序。

（2）同一个子任务的有关操作不要分散在不同地方，例如，变量 A，B，C，D 分别在四个模块中用到，不要用一个语句一次读入。

```
    10  READ  A，B，C，D
或：10  INPUT  A，B，C，D
```

而应分别在各模块中读入，将读数据与对这些数据的操作放在一起，以便于阅读，而不致在读程序时上下来回寻找有关变量。

（3）DATA 语句应集中在一起，一般放在程序的最后部分（END 语句之前）。

（4）变量名应尽可能采取“见名知意”的原则。例如以 RATE 代表利率，COUNT 代表计数，PAY 代表工资等。特别要注意不要使用含义相反的变量名（如用 DENOM（denominator 的缩写）代表“分子”），用 NUMER（numerator 的缩写）代表分母）。

（5）不要将字母和数字交叉出现在变量名中。例如 A0SS，使人不易辨别第二个字符是字母 O 还是数字 0，等三、四个字符是 5 还是 S。又如 A1B7 容易误认为 AIBT，敲入程序时容易出错，且不致检查。

（6）要善于利用注解语句为程序和各程序段作说明。

（7）如果所用的计算机系统能用汉字，应当尽量在 REM 语句中和字符串中使用汉字，用

汉字输出信息。

(8) 用 INPUT 语句输入数据时应当用“提示信息”。如：

```
10  INPUT “请输入学生姓名和学号”, NAME, NUM
```

(9) 对输入数据的合法性进行检查，以保证数据的正确性。

(10) 在一个程序或一个子程序中，在两个程序段中用空语句或注释语句分隔开，以便于阅读（相当于文章的分段）。如：

```
10
 ⋮   (第一个程序段)
100  REM
 ⋮   (第二个程序段)
200  REM
     (第三个程序段)
```

100 语句和 200 语句可以只写“REM”而不加文字注释，目的只在起一空行作用，以分隔两个程序段。也可以在 100 语句和 200 语句中不写 REM，只写语句标号，在 LIST 时，该行只有标号（即空语句）。

(11) 采取缩进格式。在循环中将循环体语句向右缩进若干格。有循环嵌套时，同一层的循环体写在同一列上，这样层次分明。

(12) 已知循环次数的循环用 FOR—NEXT 循环，不知循环次数而只给出循环条件的，用 WHILE 循环。不要从循环体中转移到循环体外（非正常出口）。

(13) 不要用多层的 IF-THEN-ELSE IF-THEN-ELSE……。因为 BASIC 只能将它们写在一行上，不易辨别其配对关系。不要在嵌套的内层 IF 语句中省略 ELSE 部分，以免使 ELSE 与 THEN 配对关系出错，

(14) 对多次重复出现的表达式，可以先求出其值，放在一个变量中，以减少重复计算。如果表达式中包含变量（变量的值不是固定的），则可用自定义函数定义该表达式为一函数。

(15) 循环变量用整型变量。

(16) 利用文件结束标志（EOF）结束输入，而不用计数来结束输入，如：

```
100  WHILE  NOT  EOF (1)
110    INPUT  #1, NAME$, ADDR$
120    IF  NAME$ = "ZHANG"  THEN  PRINT  NAME$, ADDR$
200  WEND
```

不必事先了解文件中有多少记录。

(17) 不要只是简单地输出计算结果，而应加以必要的文字说明，输出的信息应是不熟悉计算机的人也能看懂。最好能做到将打印的结果不必经过再加工就能成为正式文件保存或上送有关部门。应该事先了解用户对输出报表的要求，必要时，应打印成表格形式，还应该有标题、栏目、说明等。

(18) 对复杂的有可供选择的多功能应用程序，最好采用“菜单技术”，以方便使用。

(19) 变量名不要与语句定义符或标准函数名相同，也不要包含它们，例如不要用 PRINTA 作变量名，如果输入时不小心插入一个空格成了 PRINT A，成了输出变量 A 了。

(20) 一个程序应当配有必要的文字说明，即程序说明书，包括题目要求、采用的数学模

型、算法、流程图、变量含义、对计算机和输入输出设备的要求、要求输入的数据等。也就是要有完整的程序文档，以便保存备查，有些程序在编制完成并使用后，发现问题想修改时往往难以理解原来程序的含义，难以下手，程序维护的代价很大。在编写程序时应当想到以后几个阶段可能遇到的问题，要“为别人着想”，“为以后着想”。

总之，一个风格好的程序要注意：简明朴实，不卖弄小技巧，不要光考虑自己，而要首先考虑别人在阅读和使用时的方便，把程序写得好懂些，再好懂些！好用些，更好用些！

## §12.4 链接与覆盖

一个大型程序，往往是由若干个功能块所组成，这些功能块可以通过链接技术，把它们“串”在一起而成为一个大型的程序，这就是链接技术在程序模块化设计技术中的应用。所谓“链接”（chain）指的是把模块化程序的各模块链接成一个可供运行的整体。

链接技术的另外一个作用是，它包含着覆盖技术，所谓覆盖（overlay）是指在运行程序过程中，不同程序模块可以共占同一模内存空间，利用覆盖技术可以在较小的内存空间中运行较大的程序。下面我们介绍的链接语句，就是按被链接程序对当前程序覆盖面的程度来分类的。

### 12.4.1 全覆盖连接语句

执行链接语句以后，内存中不保留原来的程序，而由被链接程序取而代之，此时可用全覆盖链接语句。它的一般形式是：

**CHAIN“＜文件名＞”［，＜行号＞］［，＜ALL＞］**

其中，＜文件名＞为被链接程序模块的文件名；

＜行号＞为被链接程序中的某一行号，意思是一旦链接成功，当前程序块消失，被链接程序调入内存，并从给定的行号开始执行。当＜行号＞省略时，程序从被链程序的起始行开始执行。

＜ALL＞这个参数，表示把当前程序运行后的所有变量都传递给被链接程序使用，这是有参链接的一种形式。当＜ALL＞这个参数省略时，可分两种情况来讨论：

（一）．程序中有 COMMON 语句。这是有参链接的另一种形式，这种形式是把需要传送到被链程序中去的变量写在 COMMON 语句的后面，而其他变量不传送。COMMON 语句的一般形式是：

COMMON 变量表

变量表中，变量之间用逗号隔开，若有数组名时，其后一定要用空括号，所有被传递的数组都不必在被链接的程序中再用 DIM 说明语句说明其空间的大小。例如

```
10    COMMON  A，DE()，C$
20    CHAIN “PROG2”
```

表示仅把变量 A，数组 DE 和字符串变量 C$ 传送到被链程序 PROG2 中去，20 行语句中，链接语句后面的＜行号＞省略，说明连链成功后，程序从被链程序的起始行继续执行。

（二）．程序中无 COMMON 语句。CHAIN 语句中无＜ALL＞选择参数，又无 COMMON 语

句相配合，那么，这种情况就叫无参链接，例如：

```
200    CHAIN "B: SQUARE"
```

执行此语句以后，当前原程序文件从内存中消失，同时清除为它而开设的所有缓冲区和关闭当前所有文件及删除所有当前的变量和数组，而后把B盘中的程序“SQUARE”调入内存，并从它的起始行开始执行。

下面我们纯粹是从熟悉链接语句的角度来举两个应用例子，例12.1是全覆盖无参链接语句的应用；例12.2是全覆盖有参链接语句的应用。

【例 12.1】已知在磁盘中有程序“MAIN”和“AAA”，读这两个程序可知，它们是互链的，首先是“MAIN”链接“AAA”，而后是“AAA”又链接“MAIN”。程序在不同的阶段，显示出不同的信息。“MAIN”和“AAA”两程序的清单如下，同时也列出了运行“MAIN”程序的结果。

```
10    ' main
20    PRINT  " a="; 1000
30    IF  INKEY $ ="" THEN 30
40    CHAIN" aaa"
50    PRINT" b="; 3000
60    END

10    ' aaa
20    PRINT" c="; 2000;" <=====' aaa'"
30    IF INKEY $ ="" THEN 30
40    CHAIN " main", 50
50    END

RUN
a=1000
c=2000 <=====' aaa'
b=3000
```

运行程序时，读者可以注意到在执行链接语句时，相应驱动器上的红灯会亮，说明此时系统正把被链程序调入内存，而原先的程序消失。

【例 12. 2】 已知磁盘中有程序“MAIN1”和“AAA1”，读这两个程序可知它们是互链的，“MAIN1”链接“AAA1”是有参链接，而且是全部参数的传送，即“MAIN1”的全部参数在链接过程中都传送给文件“AAA1”。而程序“AAA1”中的链接语句本身没有<ALL>参数，参数的传送是通过COMMON语句进行的，例中把一个参数D传送给被链程序“MAIN1”，其他参数不传送，链接成功以后，“AAA1”程序从内存中又消失，系统把“MAIN1”程序再次调入内存，并从指定的行号（70语句）开始执行直到结束。程序MAIN1和“AAA1”以及运行程序“MAIN1”以后的结果如下：

```
10    ' main1
20    A=10: B=20: c=30
30    PRINT" a+b+c="; A+B+C
60    CHAIN "aaa1",, ALL
```

```
70    PRINT "c+d="; C+D
80    END

10    ' aaa1
20    PRINT  "b-a="; B-A; " (======' aaa1'"
30    PRINT  "c-a="; C-A; " (======' aaa"
40    D=40; E=50
50    PRINT  "e-d="; E-D; " (======' aaa"
60    COMMON  D
70    CHAIN  "main1", 70
80    END

RUN
a+b+c=  60
b-a=10   (======' aaa1'
c-a=20   (======' aaa1'
e-d=10   (======' aaa1'
c+d=40
```

### 12.4.2 全不覆盖(合并)链接语句

执行链接语句以后，若同时需要保留全部当前程序，那么可应用全不覆盖（即合并）的链接语句，它的一般形式是：

**CHAIN MERGE "<文件名>" [，<行号>] [，<ALL>]**

程序执行此语句时，相应的驱动器红灯亮，此时系统把被链程序调入内存并与当前程序合并。链接成功以后，程序从被链程序的<行号>继续执行。其中<行号>和<ALL>两参数的含义跟全覆盖链接语句相同。

有 MERGE 关键字的链接语句（包括下面要谈的部分覆盖链接语句），在使用时要注意：

(1)、被链接程序的行号与原来程序的行号不能重复，否则，被链程序行号的内容将代替原来程序相应的行号内容。(原来程序也称当前程序)。

(2)、要求被链程序必须是 ASCII 码文件。

### 12.4.3 部分覆盖链接语句

当链接以后，要求保留原来程序的一部分，其余部分都删除，那么可以用部分覆盖链接语句，它的一般形式是：

**CHAIN MERGE "<文件名>" [，<行号>] [，<ALL>]，DELETE<范围>**

其中，<范围>指的是当前程序中要求被删除的语句范围。所以，<范围>这个参数在实际应用时应写成<行号 1>—<行号 2>。

程序执行此语句时，首先删除当前程序中<行号 1>到<行号 2>的语句内容，而后把被链程序调入内存与当前程序的留下部分合并，合并以后，程序从被链程序的那个指定行号继

续执行。

下面举一个应用部分覆盖链接语句的例子。

**【例 12. 3】** 在人口普查中，常需要分析各项调查数据，而这些数据是通过每人填写一张调查表来获得的，调查表的格式如表12.1所示。

表 12. 1

| 调　查　表 | | |
|---|---|---|
| 性　别 | 年　龄　组 | |
| □1. 男性 | □1. <20 | □4. 41—50 |
| □2. 女性 | □2. 21—30 | □5. 51—60 |
| | □3. 31—40 | □6. >60 |

表中，性别用数字 1 代表男性，2 代表女性；而年龄用 1、2、3、4、5、6 分别代表小于 20 岁、21～30 岁、31～40 岁、41～50 岁、51～60 岁、大于 60 岁六个年龄等级。本程序统计 20 人，分主模块和子模块，其中主模块完成统计工作，例中从 DATA 语句中读入数据（其实在统计千万人的数据输入时，用卡片输入机比较方便）。子模块完成制表工作。

（1）主模块程序

用数组 D（2，6）来存放不同性别和年龄层次的人数。其中，数组元素 $D_{11}$～$D_{16}$ 用来统计男性各年龄组的人数；数组元素 $D_{21}$～$D_{26}$ 用来统计女性各年龄组的人数；$D_{01}$～$D_{06}$ 用来统计男女性合在一起的各年龄组的人数；$D_{10}$ 统计男性总数；$D_{20}$ 统计女性总数；$D_{00}$ 统计人口总数。

程序清单如下：

```
10    ' chain
20    N=2: M=6
30    DIM D (N, M), L$ (N), C$ (M)
40    FOR I=0 TO N: FOR J=0 TO M: D (I, J) =0: NEXT J, I
50    READ  SEX, AGE: IF SEX=0 THEN 90
60    D (SEX, AGE) =D (SEX, AGE) +1: D (0, 0) =D (0, 0) +1
80    GOTO 50
90    FOR I=0 TO N: READ L$ (I): NEXT
100   FOR J=0 TO M: READ C$ (J): NEXT
110   TITLE$ "sex/age analysis"
190   CHAIN MERGE "report", 200, ALL, DELETE 200—990
200   DATA 2, 1, 1, 2, 2, 3, 1, 1, 2, 5, 2, 6, 2, 1, 1, 2, 2, 1, 1, 1
210   DATA 1, 2, 2, 2, 1, 3, 1, 3, 2, 3, 1, 3, 2, 3, 1, 4, 1, 5, 1, 3
970   DATA 0, 0
980   DATA totle, male, female
990   DATA totle, <20, 21—30, 31—40, 41—50, 51—60, >60
```

其中，30 语句定义数组，D 数组统计人数，L$ 数组放表格的纵向标题，C$ 数组放表格的横向标题；50～80 行读数据（性别和年龄），数据的结束标志是 0；90～100 行读表格的纵、横标题；110 行给表格的总标题赋值，190 行是部分覆盖链接语句，是全部参数的传送，一旦链接成功，删除当前程序的 200～990 行语句，调入被链程序”REPORT”与当前程序的留下部

分合并。

(2) 子模块程序

子模块程序是制表程序，运用制表语句、格式输出语句以及一些专用字符，把表格数据画在适当的位置上。

程序请单如下：

```
200    ' report
240    PRINT TAB (30); USING "\␣␣␣␣␣␣␣␣␣␣␣␣␣␣\"; TITLE$
250    PRINT CHR$ (218) +STRING$ (71, 196) +CHR$ (191)
260    PRINT "|␣Items␣␣|";
270    FOR J=0 TO M: PRINT USING "\␣␣␣␣␣␣\|"; C$ (J);: NEXT J
281    PRINT : GOSUB 350
290    FOR I=0 TO N
300         PRINT USING "|\␣␣␣␣␣␣\"; L$ (I);
310         FOR J=0 TO M: PRINT USING "␣␣####␣␣|"; D (I, J);: NEXT J
312         IF I=N THEN 335
320         PRINT: GOSUB 350
330    NEXT I
335    PRINT: PRINT CHR$ (192) +STRING$ (71, 196) +CHR$ (217)
340    END
350    PRINT CHR$ (195);
360    FOR J=0 TO M: PRINT "————————+";: NEXT
365    PRINT "————————+"
380    RETURN
```

运行主程序的结果如下：

RUN

sex/age analysis

| Items | totle | <20 | 21—30 | 31—40 | 41—50 | 51—60 | >60 |
|---|---|---|---|---|---|---|---|
| totle | 20 | 5 | 4 | 7 | 1 | 2 | 1 |
| male | 11 | 2 | 3 | 4 | 1 | 1 | 0 |
| female | 9 | 3 | 1 | 3 | 0 | 1 | 1 |

OK

在编制和运行大型程序时，链接和覆盖技术是十分有用的，它能解决“小马拉大车”的问题。

# §12.5 陷阱技术

所谓陷阱技术，简单地说是使所设计的程序在运行过程中，对一些事件有能力进行处理，处理完毕以后，程序继续工作。而这些事件的发生，可以是定时的或者是随机的。允许用户捕获的事件主要有程序性出错；利用键盘上的功能键输入；时间定时器设定时间已到以及异步通讯中断等。本节简单介绍前三种事件的处理技术。它们的一般过程都是先编好一段预想事件处理程序（或称续元处理程序），再在程序的开头放一条陷阱设置语句，指出续元程序的

入口。

### 12.5.1 出错陷阱

一般情况下，如果程序发生错误，正在运行着的程序会自动停止执行，并显示出错信息。那么，应用出错陷阱技术，可以捕获一些预想到的错误，此时程序不停止运行，而是转去处理续元程序，处理完毕以后，程序又恢复正常运行。与出错陷阱技术有关的语句是：

（1）出错陷阱设置语句：

**ON ERROR GOTO〈行号〉**

此语句可以写在程序的任何地方，运行程序以前，BASIC 要查看程序中是否有此语句，一般把它放在程序的前面，以节约查看时间。程序中，若有此语句，那么在运行程序的过程中，出现错误时，程序转去指定的行号去执行，例如：

ON ERROR GOTO 5000

如出现错误，程序转至行号为 5000 的语句行继续执行，用户应该事先编制一个错误陷阱续元程序（行号从 5000 开始），它的任务是分析错误；处理错误并通知用户；继续执行程序或等待用户新的指令。而分析错误的主要手段是利用 BASIC 中的二个变量，它们是：

**ERR** 用来取当前的错误号；

**ERL** 用来取当前的错误行号。

MS BASIC 常见的出错信息如表12.2所示。

MS BASIC **常见的错误信息表** **表** 12.2

| 错误号 | 含　义 | 错误号 | 含　义 |
|---|---|---|---|
| 1 | NEXT without FOR（没有配对的 FOR 语句） | 16 | string formuja too complex（字符串表达式太复杂） |
| 2 | Syntax error（语法错） | 24 | Device timeout（设备超时） |
| 3 | RETURN without GOSUB（没有配对的 GOSUB 语句） | 26 | FOR without NEXT（没有配对的 NEXT 语句） |
| 4 | Out of data（数据读完） | 27 | Out of paper（打印机用完或打印机未开） |
| 5 | Illegal function call（非法的函数调用） | 29 | WHILE without WEND（有 WHILE 无 WEND） |
| 6 | Overflow（数值上溢） | 50 | FIELD overflow（字段溢出） |
| 7 | Out of memory（内存溢出） | 53 | File not found（文件未找到） |
| 8 | Undefinde line number（行号未定义） | 55 | File already open（文件已经打开） |

续表

| 错误号 | 含　义 | 错误号 | 含　义 |
|---|---|---|---|
| 9 | Subscript out of range<br>（下标溢出） | 58 | File already exists<br>（文件已存在） |
| 10 | Duplicate definition<br>（多重定义） | 61 | Disk full<br>（磁盘满） |
| 11 | Division by zero<br>（除数的零） | 64 | Bad file name<br>（非法文件名） |
| 13 | Type mismatch<br>（类型不匹配） | 67 | Too many files<br>（文件太多） |
| 14 | Out of string space<br>（字符串空间溢出） | 70 | Disk Write Protect<br>（写保护盘） |
| 15 | String too long<br>（字符串太长） | 71 | Disk not ready<br>（盘未准备好） |

（2）恢复语句：

RESUME ［〈行号〉］

意思是执行了续元程序以后，恢复到指定的行号执行，当<行号>省略时，恢复到出错的行号执行。若<行号>为 NEXT，则表示恢复到出错行的下一行执行。

【例 12.4】求平方根。

程序清单和运行结果如下：

```
10    ' ERR
20    ON ERROR GOTO 70
30    INPUT "x="; X
40    IF X=9999 THEN END
50    PRINT "sqr (x) ="; SQR (X)
60    GOTO 30
70    PRINT "Imaginary root:"; SQR (-X); "i"
80    PRINT "err="; ERR," erl="; ERL
90    RESUME 30

RUN
x=? 8↙
sqr (x) =2.828427
x=? -8↙
sqr (x) =Imaginary root: 2.828427 i
err=5   erl=50
x=? 9999↙
OK
```

程序中，20 语句为出错陷阱设置语句，从 70 行开始是错误事件处理程序（即续元程序）. 当输入变量 x 值是负值时，程序转入 70 行开始执行，输出虚根，同时打印出错误号和错误所在行，而后程序恢复到 30 行语句执行。

### 12.5.2 功能键陷阱

功能键陷阱设置语句是：

ON KEY (n) GOSUB<行号>

其中，<行号>为续元程序的入口，而 n 的含义如下表所示。

| n | 1～10 | 11 | 12 | 13 | 14 | 15～20 |
|---|---|---|---|---|---|---|
| 对应功能键 | F1～F10 | ↑ | ← | → | ↓ | 自定义功能键 |

可见，n 的值 1～10 代表功能键 F1～F10，11～14 代表光标上下左右移动键，15～20 为自定义功能键。

程序中有了功能键陷阱设置语句以后，当要用到 n 号功能键中断时，还必须用“允许中断发出语句”：

KEY (n) ON

当要停止第 n 号功能键发出中断，但又要记注已发生的事件（以备使用）时，则需用“停止中断发出语句”：

KEY (n) STOP

下面举一个简单例子，介绍功能键陷阱的用法。

**【例 12.5】** 设有一程序正常运行时在屏中显示时钟（TIME $）的变化，但要求按 F1 键后在第 24 行上显示字符 TRAP。程序如下：

```
5 ' TRAP
10 CLS
20 ON KEY (1) GOSUB 500: KEY (1) ON
30 KEY (1) STOP
40 LOCATE 12, 36: PRINT TIME $
50 KEY (1) ON
60 GOTO 30
500 LOCATE 24, 1: PRINT "TRAP"
510 RETURN
```

程序中，20 行为功能键（F1）陷阱设置语句和允许中断发出语句，但为了在正常显示时钟时，不受中断干扰，程序使用停止中断发出语句（30 语句），它有记忆功能，40 行语句是正常显示，显示完毕，50 行语句立即开放 F1 键的中断，以便在此期间发生的中断得到处理。

### 12.5.3 时钟陷阱

时钟陷阱技术可用于定时数据采样，限时回答问题等环境中，时钟陷阱设置语句为：

ON TIMER (n) GOSUB<行号>

其中，n 取值 1～86400 秒（24 小时），表示定时器的设定时间值。<行号>为续元程序的起

始行号。它的设置过程是：

（1）编制续元程序；

（2）用 ON TIMER （n） GOSRB 语句设置时间间隔；

（3）用 TIMER ON 语句启动时钟计时，当时间计时与设定时间相同时，则发出时钟中断，转去续元程序执行，此时时钟归零并重新开始计时，执行好续元程序以后，系统又返回到被打断的程序处继续执行。当第二次时间计时与设定时间相同时，又发出第二次中断，这样不断重复，一直到遇到关闭时钟计时语句 TIMER OFF 为止。

**【例 12.7】** 每隔 10 秒钟在屏中央显示日期和时间，程序如下：

```
10 ' TIMER
20 INPUT " T$ mm—dd—yyyy"; T$ : DATE$ =T$
30 INPUT " S$ hh: mm: ss"; S$ : TIME$ =S$
40 CLS
50 ON TIMER (10) GOSUB 1020
60 KEY OFF
70 TIMER ON
80 IF INKEY$ = "" THEN 80
90 END
1020 LOCATE 12, 30
1030 PRINT DATE$ ; "          "; TIME$
1060 RETURN
```

程序中，20 行和 30 行的作用为对日期和时间赋初值，50 行是时钟陷阱设置语句，设置时间是 10 秒钟。续元程序从 1020 行开始；60 行“KEY OFF”的作用是将显示屏上第 25 行的“功能键作用显示”取消（通常显示屏上的第 25 行专用以显示功能键的作用，而不能用作程序显示结果，如需利用第 25 行显示运行信息，则用 KEY OFF 关闭“功能键显示”）。70 行为开启时间计时。1030 行为在指定位置显示日期和时间，TIME$ 为系统时钟函数，它以 30 行给它的初值为基础，不断改变时间值。在执行 80 行过程中，只要每到 10 秒钟就执行一次子程序，显示时间。如要终止运行按任一键即可。注意执行子程序完后返回到被打断处继续执行，而不是返回 60 行。

## 习　　题

12.1　请叙述结构化程序的主要特点。结构化程序与非结构化程序主要区别是什么？

12.2　如何实现一个结构化程序设计过程？它应经历哪几个步骤？

12.3　在进行模块分解时应遵循哪些原则？

12.4　什么是软件工程方法？软件生命周期包括哪几个阶段？结构化程序设计在软件工程中的位置和作用是什么？

12.5　BASIC 中数值数据分为哪几个类型？各有什么特点？在什么情况下使用哪个类型？

12.6　写出下面程序的运行结果.

（1）
```
10 FOR A=1 TO 10 STEP 0.5
20     B%=A : PRINT B%
```

```
30 NEXT A
40 END
```

(2)

```
10 DEFINT A : DEFDBL B
20 B1= 1/16 : B2=1/7
30 B3= 1#/6 : B4=1#/7
40 A1=B1 : A2=B2 : A3=B3 : A4=B4
50 C1=B1 : C2=B2 : C3=B3 : C4=C4
60 PRINT A1, A2, A3, A4
70 PRINT B1, B2, B3, B4
80 PRINT C1, C2, C3, C4
90 END
```

12.7 用双精度数据编程序，求一元二次方程式的根：

$$x_2 - 0.1x - 2.5 = 0$$

12.8 根据你自己的经验，你认为怎样的程序才是一个好的程序？

12.9 你能对§12.3程序风格中列举的应注意的问题作一些补充吗？你自己平时在编程序时哪些已注意到了？哪些还没注意到？

12.10 请你从本书中选择任一例题程序，对它进行修改补充，以使之具有更好的程序风格。

12.11 请叙述链接与覆盖的概念。在什么情况下会用到链接与覆盖。

12.12 将第十一章习题10.12 的要求改为：不用菜单，而用链接覆盖技术，当做完加法练习后自动链接入减法模块，然后是乘法模块、除法模块，最后计算总分（全对为400分）。

12.13 设计一个程序，包括5个模块，分别用来实现：矩阵相加、相减、相乘和矩阵打印。这5个模块分别以文件名"MATIN"，"MATADD"，"MATSUB"，"MATMUT"，"MATPRI"。主控程序先显示程序功能说明，然后链接入"MATIN"，输入数据，"MATIN"模块再链接入"MATADD"……，即第一个模块链接入第二个模块，第二个模块链接入第三个模块，在每个模块中分别完成运算和打印。第五个模块再链接主控模块"MAIN"，在MAIN模块中最后打印"The end"，然后结束。

12.14 设计一个时钟计时显示程序，每秒显示一次时间，每隔1分钟发出一次响声。用BEEP语句式PRINT CHR$（7）发出声响。如：

```
10 BEEP
```

或

```
10 PRINT CHR$ (7)
```

12.15 设计一个有陷阱的程序，用来处理"未打开打印机而用了LPRINT语句"的情况。当出现此情况时，由ON ERROR语句转到指定的陷阱续元程序，在此程序段中输出信息"Check printer"，然后等待操作人员打开打印机后按任何一键程序恢复运行。"未打开打印机"的错误代码为27。

12.16 设计一个程序，用FOR循环求 $\Sigma\frac{1}{n}$，当按功能键<F3>后，程序就停止累加，输出最后一项（$\frac{1}{n}$）的值以及 $\Sigma\frac{1}{n}$ 之值。

# 第十三章　常用算法程序举例

到目前为止，我们已介绍了使用 BASIC 语言进行程序设计的基本方法，并且举了一些较简单的例子。考虑到在学习各章过程中主要的任务是熟悉 BASIC 有关语句和函数的应用，同时考虑到不同领域、不同程度的读者的情况，在前面各章中所介绍的算法都是最基本的，即使没有学过高等数学的读者也能掌握。

在此基础上，读者应当进一步学习和掌握有关的算法。程序设计要综合用到算法、数据结构、结构化程序设计方法、以及计算机语言工具这四个方面的知识。可以用以下公式表达：

结构化程序＝算法＋数据结构＋结构化程序设计方法＋计算机语言工具

或者说，一个程序设计的过程是：根据选定的数据结构，选择合适的算法，采用结构化程序设计的方法来描述此算法，最后用一种计算机语言实现它。

BASIC 提供的数据结构比较简单，主要有简单变量和数组两大类。因此，BASIC 程序设计的重点是算法设计。算法分为数值运算的算法和非数值运算的算法两大类。数值运算一般可以借鉴现成的数学模型，可以运用数值分析方法，因此对数值运算的研究比较深入，算法比较成熟。对各种数值运算都有比较成熟的算法可供选用。常常把这些算法写成程序形式汇编成册或者存放在磁盘或磁带上，供用户调用。例如有的计算机系统提供“BASIC 数学程序库”或“FORTRAN 数学程序库”等，使用十分方便。而非数值运算的种类繁多，要求各异，难以规范化，因此只能对一些典型的非数值运算算法（如排序算法）作比较深入的研究。其它的非数值运算问题，往往需要使用者参考类似算法重新设计解决特定问题的专门算法。

在本章中不可能罗列各种算法，只能介绍一些常用的典型算法，目的是帮助读者了解如何设计一个问题的算法，并能举一反三。

本章中的程序可以结合前面各章内容进行学习。不同专业，不同程度的读者可以根据需要选学。

## §13.1　穷　举　法

有一些问题，需要从若干个方案中选择符合需要的一种或多种方案，又无现成的公式可用时，往往用“穷举法”，即把所有的可能都试一下，一个一个地判断选择是否所需的。第五章中例5.25　就是穷举法的一个例子。

**【例 13.1】** 百鸡问题。我国古代数学家张丘建在《算经》中出了一道题：“鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一。百钱买百鸡，问鸡翁、母、雏各几何？”其意为：每只公鸡 5 元，母鸡 3 元，小鸡三只 1 元。用 100 元买 100 只鸡，问公鸡、母鸡、小鸡各可买多少？

先列出数学式子。设 x、y、z 分别为公鸡、母鸡和小鸡数，则从题意可以写出：

$$\begin{cases} x+y+z=100 & (1) \\ 5x+3y+\frac{z}{3}=100 & (2) \end{cases}$$

三个未知数只有两个方程式，不能解出 $x,y,z$。如果三个未知数中已确定了一个，其它两个未知数就可以求出了。假设 $X$ 已知，则解上述方程可得：

$$\begin{cases} y=25-\frac{7}{4}*X \\ z=75+\frac{3}{4}X \end{cases}$$

但现在 $X$ 并不知道，所以用穷举法把每一个 $X$ 都试一下，看对于某一个 $X$，求出的 $y$ 和 $z$ 是否都为正整值。可编出以下程序：

```
10 PRINT "cock", "hen", "chick"
20 FOR X=0 TO 100
30   Y=25-7/4*X
40   Z=75+3/4*X
50   IF Y>0 AND Z>0 AND INT (Y) =Y AND INT (Z) =Z THEN PRINT X, Y, Z
60 NEXT X
70 END
```

其实公鸡不可能有 100 只，最多 20 只（因 5 元一只）。实际上，也不可能买 20 只，因为 100 元全买了公鸡，就买不了母鸡和小鸡了，因此 X 最大为 19。20 语句可改为："FOR X=0 TO 19"。

运行结果如下：

| cock | hen | chick |
|---|---|---|
| 0 | 25 | 75 |
| 4 | 18 | 78 |
| 8 | 11 | 81 |
| 12 | 4 | 84 |

但是这种方法要事先解方程式，要人做的事太多，不方便。可以用另一种方法；先定一个 $x$ 值，再用穷举法一一找出 $y$ 的可能值，然后看在一组 $x,y$ 值的前提下，有无 z 可以满足百钱买百鸡的条件，可以这样写出程序：

```
10 PRINT "cock", hen", "chick"
20 FOR X=0 TO 100
30   FOR Y=0 TO 100
40   Z=100-X-Y
50   IF 5*X+3*Y+Z/3=100 THEN PRINT X, Y, Z
60   NEXT Y
70 NEXT X
80 END
```

这样把所有的 $x,y,z$ 可能值的组合都一一判断是否符合要求。需要执行内循环体 101×101=10201 次。实际上，如前述 x 不可能大于 19，同理 y 不会大于 33。因此，20 和 30 语句可以改

写为：

```
20 FOR X=0 TO 19
30      FOR Y=0 TO 33
```

这样内循环体只需执行 20×34=680 次。减少 90%以上。可以看出，稍作分析，就可以改进程序，提高运行效率。

穷举法看起来好象很“笨”，要试遍所有的可能方案，但这却是“没有办法时的办法”，保证所能出结果，计算机用循环来实现穷举，速度是很快的。当然如果能用直接的方法求出结果可以不用穷举法。

## §13.2 递 推 法

在第五章中介绍了递推的概念和方法，例如求 Fibonacci 数列，求 $n!$ 等。用“递推”方法解题的关键是确定：①递推公式；②初始条件（边界条件）。

递推的一种重要形式是“倒推”，即从已知的最后结果和递推公式倒推出初始情况。

**【例 13.2】**猴子吃桃。

小猴子第一天摘下若干个桃子，当即吃掉了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后第天早上都吃了前一天剩下的一半另一个。到第 10 天早上猴子想再吃时，见到只剩一个桃子了。问第一天猴子共摘了多少个桃子。

这是一个典型的“倒推”问题。从最后一天起逐天推算出前一天的桃子数，直到推出第 1 天的为止。设第 n 天桃子数为 $x_n$，已知它是前一天的桃子数 $x_{n-1}$ 的 1/2 再减去 1。即：

$$x_n = \frac{1}{2}x_{n-1} - 1$$

即：$$x_{n-1} = (x_n + 1) \times 2$$

利用此公式可以从第 n 天的桃子数推出其前一天的桃子数。边界条件为：

$$x_{10} = 1$$

可写出以下程序：

```
10 DIM X (10)
20 X(10)=1
30 FOR N=10 TO 2 STEP -1
40   X (N-1) = (X (N) +1) *2
50 NEXT N
60 PRINT" the number of peaches is :"; X (1)
70 END
```

运行结果如下：

```
the number of peaches is ; 1534
```

程序用了数组，X（1）～X（10）的值分别为第 1 天到第 10 天的桃子数，今用递推方法，但不是迭代法，并不存在用一个变量的新值代替它的原值。下面的程序不用数组，用迭代方法来实现递推。

```
10   X=1
```

```
20   FOR N=10 TO 2 STEP -1
30       X= (X+1) *2
40   NEXT N
50   PRINT " the number of peaches is :"; x
60   END
```

注意：FOR 语句中 N 从 10 变到 2，不要写成：FOR N=10 TO 1，因为每次循环是从本天推出前一天的情况。最后一次循环是从第二天推出第一天的情况，其执行 9 次（而不是 10 次）循环。

## § 13.3 求函数的定积分

求函数 $f(x)$ 在 $[a,b]$ 区间的定积分，

即：$$\int_a^b f(x)dx$$

在几何意义上，定积分是 $f(x)$ 曲线与直线 $x=a, y=0, x=b$ 所围成的曲线梯形的面积，用计算机求定积分不是用解析法求它的准确值，而是用近似方法求面积，为了求出该面积，可以将 $[a,b]$ 分为若干个小区间，每个区间长度为 $\frac{b-a}{n}$，$n$ 为区间数。近似地求出每个小曲线梯形面积，然后把它们加起来，就近似地得到总面积，即定积分的近似值，$n$ 愈大（即区间分得愈小），愈接近准确值。见图 13.1。

近似求小曲线梯形的方法，常用的有以下三种。

(1) 矩形法。用小矩形代替小的曲线梯形（见图13.2 (*a*)）. 然后将各小矩形面积累加之。

(2) 梯形法。用小梯形代替小的曲线梯形（见图13.2 (*b*)）。

(3) 辛普生（*Sipson*）法，在小区间范围内，用一条抛物线 $f_1(x)$ 代替 $f(x)$。$f_1(x)$ 与 $f(x)$ 在 $c, d$ 和 $\frac{c+d}{2}$ 三点上的函数值相等。求出该抛物线 $f_1(x)$ 和直线 $x=c, y=0, x=d$ 所围的小曲线梯形面积。

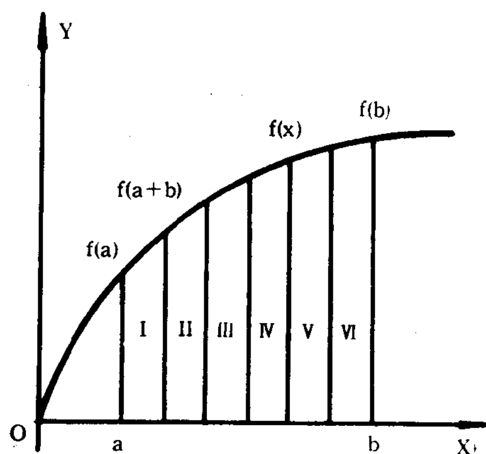



图 13. 1

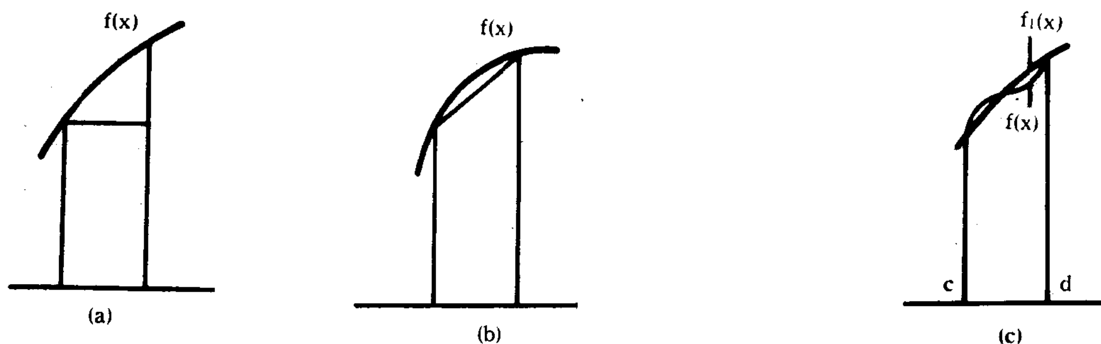



图 13.2

## 13.3.1 矩形法

矩形的面积为底×高。第一个小矩形的底为 $f(a)$，高为 $h=\frac{b-a}{n}$，即小区间宽。第二个小矩形的底为 $f(a+h)$，高仍为 h，……，最后一个小矩形的底为 $f(a+(n-1)\cdot h)$。

第 $i$ 个矩形面积等于：

$$S_i = h\cdot f(a+(i-1)\cdot h)$$

用矩形法求定积分的 N－S 图见图 13.3

**【例 13. 3】** 求 $\int_0^1 sinxdx$

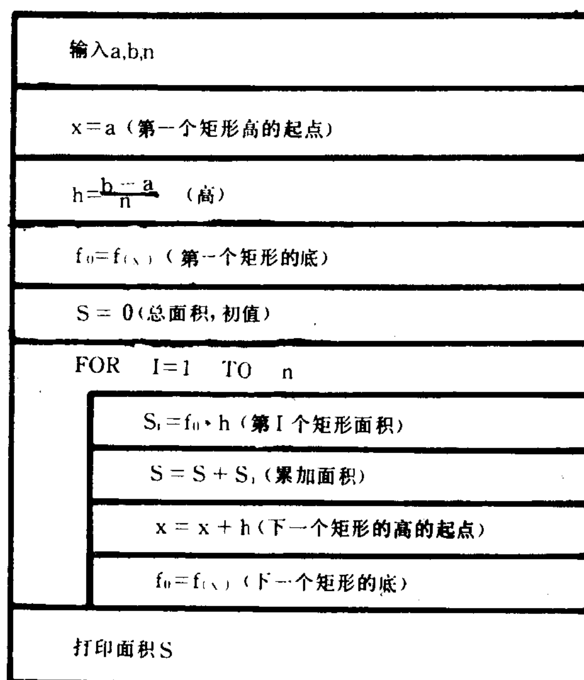



图 13.3

根据流程图编出程序：

```
10  INPUT" A, B, N="; A, B, N
```

```
20  X=A
30  H= (B—A) /N
40  F0=SIN (X)
50  S=0
60  FOR I=1 TO N
70    SI=F0 * H
80    S=S+SI
90    X=X+H
100    F0=SIN (X)
110  NEXT I
120  PRINT "S="; S
130  END
```

以下是三次运行的结果：

①A，B，N=0，1，10↙

S= .417241

②A，B，N=0，1，100↙

S= .4554864

③A，B，N=0，1，1000↙

S= .4592744

当然也可以将第一个小矩形的底定为 $f(a+b)$，此时小矩形面积比实际的曲线梯形面积大，只需将 40 语句改为：

40 F0=SIN (X+H)

即可。从程序运行结果可见，区间数 N 愈大，就愈接近积分的准确值，但运行时间也相应加长。

如果欲求其它函数的定积分，只需修改 40 和 100 语句即可。也可用自定义函数：

5 DEF FNF(X)=SIN(X)

然后将 40 和 100 语句中的 SIN (X) 改成 FNF (X) 即可。

### 13.3.2 梯形法

梯形的面积是：$\frac{\text{上底}+\text{下底}}{2}\times$高

第一个小梯形面积为：

$$s_1=\frac{f(a)+f(a+h)}{2}\times h$$

第 i 个小梯形面积为：

$$s_i=\frac{f(a+(i-1)\cdot h)+f(a+i\cdot h)}{2}\times h$$

**【例 13. 4】**用梯形法求 $\int_0^1 \sin x dx$

程序如下：

```
10    INPUT " A, B, N=", A, B, N
```

```
15    H= (B-A) /N
20    X=A
40    S=0
60    FOR I=1 TO N
70      SI=(SIN((I-1)*H)+SIN(I*H))*H/2
80      S=S+SI
110   NEXT I
120   PRINT "S="; S
130   END
```

以下是三次运行的结果：

① A，B，N=0，1，10

S= .4593146

② A，B，N=0，1，100

S= .4595938

③ A，B，N=0，1，1000

S= .4596978

其实，上一个小梯形的下底就是下一个小梯形的上底，因此不必每次重复计算。程序可改写如下：

```
10    INPUT "A，B，N="，A，B，N
15    H= (B-A) /N
40    S=0
50    F1=SIN (A)        (上底)
60      FOR I=1 TO N
70      F2=SIN (A+I*H)（下底）
75      SI= (F1+F2) *H/2（小梯形面积）
80      S=S+SI（小梯形面积累加）
85      F1=F2（将本次下底作为下一个梯形的上底）
110   NEXT I
120   RPINT "S="; S
130   END
```

以上是一次求出一个小梯形面积，几何意义清楚，易于理解。也可以先找出 n 个小梯形面积的代数和的公式，然后据此编程序。

设 $f_0,f_1,f_2\cdots f_n$ 分别为 $x$ 等于 $x_0,x_1,x_2,\cdots x_n$ 时函数 $f(x)$ 的值。

$$\int_a^b f(x)dx \approx \frac{h}{2}(f_0+f_1)+\frac{h}{2}(f_1+f_2)+\frac{h}{2}(f_2+f_3)+\cdots+\frac{h}{2}(f_{n-1}+f_n)$$

$$=\frac{h}{2}[f_0+2(f_1+f_2+\cdots f_{n-1})+f_n]=\frac{h}{2}[f_0+2(f_1+f_2+\cdots+f_n)-f_n]$$

如果 $f(x)=SIN(X)$，则：

$f_0=sinx_0=sin(a)$， $f_1=sinx_1sin(a+h)$， $f_2=sinx_2=sin(a+2h),\cdots$

$f_n=sinx_n=sin(a+nh)=sinb$

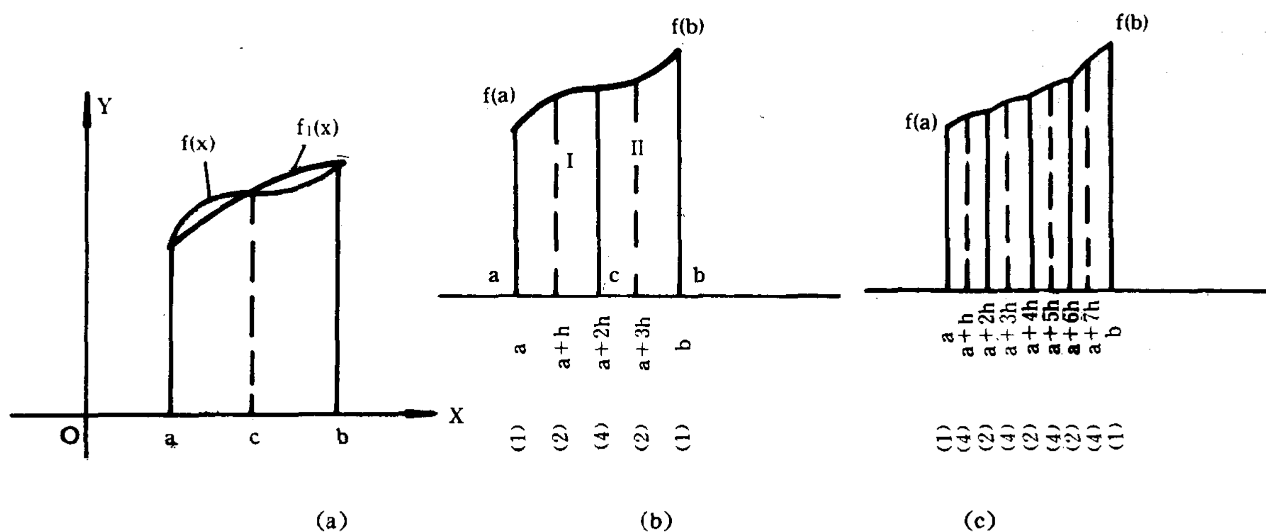
据此可编写程序：

```
10    INPUT "A，B，N="; A，B，N
```

```
20    H= (B-A) /N
30    F0=SIN(A)
40    S=F0
50    FOR I=1 TO N
60      F=SIN (A+I*H)
70      S=S+2*F
80    NEXT I
90    S= (S-SIN (B)) *H/2
100   PRINT "S="; S
110   END
```

## 13.3.3 辛普生法

在小区 [a，b] 间用一抛物线 $f_1(x)$ 代替原曲线 $f(x)$ 。抛物线 $f_1(x)$ 是怎样决定的呢？取 $a,b$ 的中点 $c$，$c$ 的坐标为 $(a+\frac{b-a}{2},0)$，通过 $c$ 点可求出 $f(c)$ 。在几何上就是作一条 $x=c$ 的直线与 $f(x)$ 交于 $f(c)$ 。通过 $f(a),f(c),f(b)$ 三点可以作出唯一的一条抛物线。见图13.4.(a)由数学知识可知：在 $[a,b]$ 区间的 $f_1(x)$ 可以由下式表示：




(a)　　(b)　　(c)

图 13. 4

$$f_1(x)=\frac{(x-c)(x-b)}{(a-c)(a-b)}\cdot f(a)+\frac{(x-a)(x-c)}{(b-a)(b-c)}\cdot f(b)+\frac{(x-a)(x-b)}{(c-a)(c-b)}\cdot f(c)$$

从上式可知，$x=a$ 时，$f_1(a)=f(a)$，$x=b$ 时，$f_1(b)=f(b)$，同理，$f_1f(c)=f(c)$。

$\int_a^b f(x)dx$ 可以近似化用 $\int_a^b f_1(x)dx$ 代替。根据抛物线定积分求值公式，有：

$$\int_a^b f_1(x)dx=\frac{h}{3}[f(a)+4f(c)+f(b)]$$

其中 $h=\frac{b-a}{2}$

还可以将 $[a,b]$ 分成两个小区间 $[a,c]$ 和 $[c,b]$。对每一个小区间分别各以一条抛物线代替原来的 $f(x)$，分别计算出两个小的曲线梯形 I 和 II 的面积（图13.4 (b)），然后把它们相加。

$$s = \int_a^b f(x)dx \approx s_1 + s_2 = \frac{h}{3}\{[f(a) + 4f(a+h) + f(a+2h)] + [f(a+2h) + 4f(a+3h) + f(b)]\}$$
$$= \frac{h}{3}\{f(a) + f(b) + 4[f(a+h) + f(a+3h)] + 2f(a+2h)\}$$

式中 $h = \frac{c-a}{2}$。从上式可以看到一条规律：看 $(a+nh)$，当 $n$ 为奇数时，它前面的系数是 4，即：$4[f(a+h) + f(a+3h)]$，$n$ 为偶数时，系数为 2，即 $2f(a+2h)$。见图13.4(b)下部括号内的数字（表示系数）。

为使求定积分更准确，再分小区间，即分为 4 个小区间：$(a,a+2h)$，$(a+2h,a+4h)$，$(a+4h,a+6h)$，$(a+6h,b)$。4 个区间的中点分别为 $a+h,a+3h,a+5h,a+7h$。$h = \frac{c-a}{4} = \frac{b-a}{8}$。见图13.4 (c)。

总面积为：四个小曲线梯形之和：

$$s \approx \frac{h}{3}\{[f(a) + 4f(a+h) + f(a+2h)] + [f(a+2h) + 4f(a+3h) + f(a+4h)] + [f(a+4h) + 4f(a+5h) + f(a+6h)] + [f(a+6h) + 4f(a+7h) + f(b)]$$
$$= \frac{h}{3}\{f(a) + f(b) + 4[f(a+h) + f(a+3h) + f(a+5h) + f(a+7h)] + 2[f(a+2h) + f(a+4h) + f(a+6h)]\}$$

如果分成 n 个小区间，则：

$$s \approx \frac{h}{3}\{f(a) + f(b) + 4[f(a+h) + f(a+3h) + \cdots + f(a+(2n-1)\cdot h)] + 2[f(a+2h) + f(a+4h) + \cdots + f(a+(2n-2)\cdot h)]\}$$

**【例 13.5】** 用辛普生法求 $\int_a^b sinxdx$

```
10   INPUT "A, B, N="; A, B, N
20   H= (B-A) / (2*N)
30   S=0
40   FA=SIN (A)
50   FB=SIN (B)
60   X=A+H
70   F2=0
80   F4=SIN (X)
```

```
90     FOR I=1 TO N-1
100      X=X+H
110      FZ=FZ+SIN (X)
120      X=X+H
130      F4=F4+SIN (X)
140    NEXT I
150    S=H/3 * (FA+FB+4 * F4+Z * F2)
160    PRINT "S="; S
170    END
```

以下是三次运行的情况：

① A, B, N=0, 1, 10↙
S= .4596978

② A, B, N=0, 1, 100↙
S= .4596975

③ A, B, N=0, 1, 1000↙
S= .4596993

说明：①$H=(B-A)/(2*N)$。当 $N=4$ 时，$H=\frac{B-A}{8}$。

② $F2$ 代表将要乘以 2 的那个多项式，F4 代表将要乘以 4 的多项式。$F2$ 的第一项应是 $f(a+2h)$，因此，可以将 $F2$ 的初值定为 0，然后在第一次循环中加 $f(a+2b)$。$F4$ 的第一项是 $f(a+h)$，因此，可将 $F4$ 的初值定为 $SIN(X)$，此时，$X$ 已被赋值为 $a+h$。$F4$ 的初值为 $f(a+h)$.

③在第一次循环中，执行第一个"$X=X+H$"语句后，$X$ 的值等于 $A+2*H$，执行第二个 $X=X+H$ 语句后，$X$ 的值等于 $A+3*H$。见表13.1。

**表** 13.1

| 第几次循环 | 执行第一个"X=X+H"后X的值 | F2的值 | 执行第二个"X=X+H"后X的值 | F4的值 |
|---|---|---|---|---|
| 1 | a+2h | f(a+2h) | a+3h | f(a+h)+f(a+3h) |
| 2 | a+4h | f(a+2h)+f(a+4h) | a+5h | f(a+h)+f(a+3h)+f(a+5h) |
| 3 | a+6h | f(a+2h)+f(a+4h)+f(a+6h) | a+7h | f(a+h)+f(a+3h)+f(a+5h)+f(a+7h) |

在三种求定积分值的方法中，矩形法的误差较大，梯形法次之，辛普生法最好。我们现在介绍的是定步长（区间）的辛普生法，还有变步长的辛普生法（可以根据需要不断分小区间直到满足误差要求为止），读者可参考有关书籍。

## §13.4 求一元方程的近似根

一元方程的根有些可以通过解析法求得，如 $x^2-2x+1=0$ 可分解为 $(x-1)^2=0$，根 $x=1$。但是在许多情况下无法分解，也就是说无法用解析法求 $x$ 的准确值。可以用近似方法求根的近似值。下面介绍几种常用的方法。

### 13.4.1 迭代法

其基本方法如下：

(1)将 $f(x)$ 写成求 $x$ 的式子：$x = g(x)$ 形式。

(2)大致估计出一个根 $x$ 的范围，给 $x$ 定一个初值 $x_0$，把它代入 $g(x)$ 中，求出 $x$ 的第一次近似值 $x_1$。

(3)再将 $x_1$ 代入 $g(x)$，即由 $g(x)$ 求出 $x_2$。再由 $g(x_2)$ 求出 $x_3$，由 $g(x_3)$ 求出 $x_4$，…如此一次一次地将求出的新值作为下次的初值代入 $g(x)$，求出 $x$ 的下一个近似值，直到前后二次求出的 $x$ 值很接近，即 $|x_{n-1} - x_n| \leqslant \varepsilon$ 为止。$\varepsilon$ 是一个给定的很小的数。这时 $x_{n+1}$ 就是所求的近似值。

**【例 13．6】** 用迭代法求方程的根

$$f(x) = x + e^x = 0$$

先找出 $x = g(x)$ 式子，可得到：

$$x = - e^x$$

估计在 0 附近有一个根。设 $x_0 = 0$。

程序如下：

```
10    DEF FNA (X) =-EXP (X)
20    X0=0
30    X1=FNA (X0)
40    WHILE ABS (X1-X0) >. 00001
50      X0=X1
60      X1=FNA (X0)
70    WEND
80    PRINT " x="; X1
90    END
RUN
x=-. 5671408
```

若经过许多次迭代后仍不收敛，就可能是发散的，为防止无限止地循环下去，可以设定最多循环次数，例如循环 50 次仍不收敛就不再迭代，使程序结束。请读者自己修改程序。

### 13．4．2 牛顿迭代法

求 $f(x) = 0$ 在 $x_0$ 附近的一个实根的方法如下：

(1) 选一个接近于 $x$（真实根）的近似根 $x_1$；

(2) 通过 $x_1$ 求出 $f(x_1)$。在几何上就是作 $x = x_1$ 交 $f(x)$ 于 $f(x_1)$。

(3) 过 $f(x_1)$ 作 $f(x)$ 的切线，交 $x$ 轴于 $x_2$。

(4) 通过 $x_2$ 求出 $f(x_2)$。

(5) 再过 $f(x_2)$ 作 $f(x)$ 的切线，交 $x$ 轴于 $x_3$。

(6) 再通过 $x_3$ 求出 $f(x_3)$，并重复以上步骤，直到前后二次求出的根之差的绝对值 $|x_{n+1}$

$-x_n| \leqslant \varepsilon$ 为止，此时认为 $x_{n+1}$ 足够接近于真实的根。

以上见图13.5。

从图中几何关系可以看出：

$$f'(x_1) = \frac{f(x_1)}{x_1 - x_2}$$

故：$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

同理 $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

这就是牛顿迭代公式，即从根的前一个近似值可以推出下根的下一个近似值。

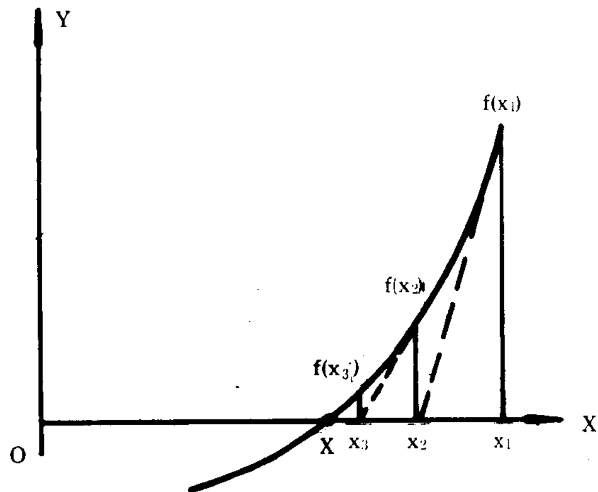



图 13.5

**【例 13.7】** 用牛顿迭代法求

$$x + e^x = 0$$

在 $x = 0$ 附近的一个实根。

按上述方法可画出 N－S 流程图。（见图 13.6）。

今：$f(x) = x + e^x$

$f'(x) = 1 - e^x$

思路是：用 $x_1$ 代表 $x$ 的第一次近似值，$F1$ 代表函数 $f(x_1)$ 的值，$D1$ 代表 $f'(x1)$ 的值，根据 $x_2 = x_1 - \dfrac{f(x1)}{f'(x1)}$ 公式求出 $x2$，即 $x$ 的下一个近似值。$N$ 代表迭代次数，每迭代一次打印一次 $N$，$x1, x2$。然后将新求出的 $x2$ 作为下一次迭式的初值 $x1$，再求出新的 $x2$，……，这样做的目的是只用两个变量 $x1, x2$ 代表了所有的近似值 $x_1, x_2, x_3, \cdots x_n$。用迭代方法处理是很方便的。每一次循环进行一次迭代，$x_1, x_2$ 在每次循环中都得到新的值。

程序如下：

```
10    DEF FNA (X) =X+EXP (X)
20    DEF FNB (X) =1+EXP (X)
30    INPUT X1
40    PRINT " n"," x1"," x2"
50    N=1
60      F1=FNA (X1)
70      D1=FNB (X1)
80      X2=X1-F1/D1
90      PRINT N, X1, X2
100      X1=X2
110      N=N+1
120      IF ABS (X2-X1) >.0000001 GOTO 60
130    END
RUN
? 1↙
n              x1              x2
1              1               0
```

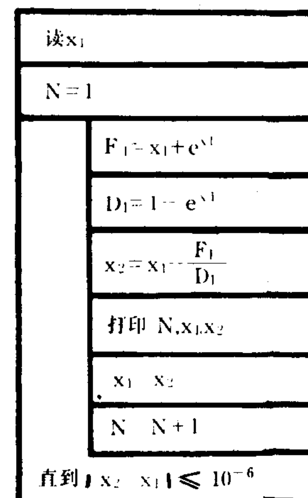



图 13.6

发现程序只执行了一次循环就结束了。原因是在 100 语句赋值之后，$x_1$ 等于 $x_2$，因此 120 语句的 $ABS(x2-x1)$ 的值为零。这个错误不易被发现。为了解决这个问题，在开始输入 $x$ 的第一个近似值不是给 $x1$ 而是给 $x_2$，然后执行赋值语句"$x1=x2$"，再根据 $x1$ 求出新的 $x2$。此时 $x_1$ 和 $x_2$ 一般是不相等的。程序改为：

```
10    DEF FNA (X) =X+EXP (X)
20    DEF FNB (X) =1+EXP (X)
30    INPUT X2
40    PRINT "n","x1","x2"
50    N=1
60    X1=X2
70    F1=FNB (X1)
80    D1=FNB (X1)
90    X2=X1-F1/D1
100   PRINT N, X1, X2
110   N=N+1
120   IF ABS (X2-X1) >.0000001 GOTO 60
130   END
RUN
? 1↙
n          x1           2
1          2            0
2          0            -.5
3          -.5          -.566311
4          -.566311     -.5671431
5          -.5674131    -.5671433
6          -.5671433    -.5671433
```

可知结果是正确的。读者应学习和掌握在调试程序时如何发现和修正错误。

## 13.4.3 弦截法

若 $f(x)$ 为单调函数（单调增或单调减），且 $f(x_1)$ 和 $f(x_2)$ 异号，则在 $x_1$ 与 $x_2$ 之间必有根。见图 13.7。弦截法的基本方法是：联接 $f(x_1)$ 和 $f(x_2)$ 二点，此直线（即弦）交 $x$ 轴于 $x_0$，测试 $f(x_0)$ 与 $f(x_1)$ 是否同符号，如果相同（如图所示那样），则表示 $(x_1,x_0)$ 之间无根，根必在 $(x_0,x_2)$ 之间。此时将 $x_0$ 作为新的 $x_1$（图中以 $x_1'$ 表示），再将 $f(x_1')$ 与 $f(x_2)$ 联结，联线交 $x$ 轴于 $x_0'$，再判断 $f(x_0')$ 与 $f(x_1')$ 是否同号…。每次缩小区间。若第一次 $f(x_0)$ 与 $f(x_1)$ 异号，则说明 $(x_1,x_0)$ 间有根，此时将 $x_0$ 作为新的 $x_2$（即把区间 $(x_0,x_2)$ 舍掉），接着在 $(x_1,x_0)$ 中用上法不断缩小区间。当区间不断缩小，直到 $|x_1-x_2|\leqslant\varepsilon$ 为止（$\varepsilon$ 为一个指定的很小的数）。此时将 $\frac{x_1+x_2}{2}$ 作为近似根值。

求 $x_0$ 的值的公式为：

$$x_0 = x_2 - \frac{x_2 - x_1}{f(x_2) - f(x_1)} \cdot f(x_2)$$

这可从图 13. 7 中相似的三角形的关系中看出以上关系。

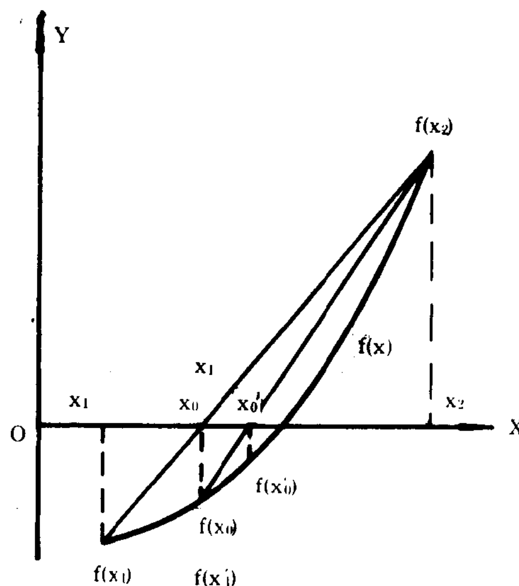



图 13. 7

**【例13. 8】** 用弦截法求下面方程的根

$$f(x) = x + e^x = 0$$

先根据上述方法画出 $N-S$ 图。（图 13. 8）。

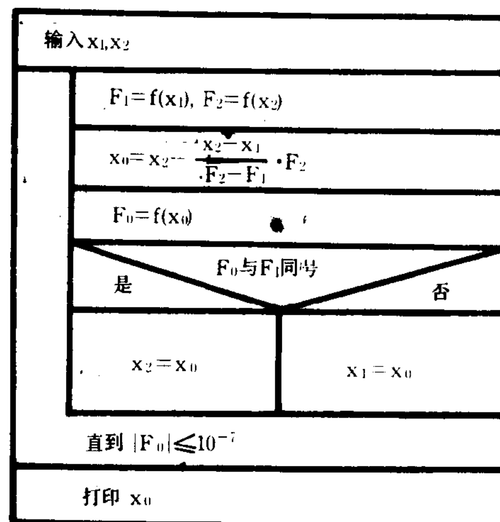



图 13. 8

还有两个问题要考虑：(1) 要对输入的 $x_1$ 和 $x_2$ 作检查，看 $f(x_1)$ 和 $f(x_2)$ 是否异号，如是，则 $(x_1,x_2)$ 间有根，否则重新输入 $x_1,x_2$，直到 $f(x_1)$ 和 $f(x_2)$ 异号为止。(2) 为防止循环很多次仍得不到结果，设定最高循环次数，用 $N$ 累计已循环次数，当 $N \geqslant 50$ 次就不再循环。因此，程序在流程图的基础上又有了发展。请注意分析。

```
10  DEF FNF (X) =X+EXP (X)
20  INPUT  "x1, x2=", X1, X2
30  IF SGN (FNF(X1)) =SGN (FNF (X2)) GOTO 20
35  N=0
40  F1=FNF(X1): F2=FNF(X2)
50  X0=X2- (X2-X1) / (F2-F1) * F2
60  F0=FNF (X0)
80  IF SGN (F0) <>SGN (F1) THEN X2=X0
      ELSE X1=X0
85  N=N+1
87  PRINT N, X1, X2
90  IF ABS (F0) >0.0000001 AND N<50  GOTO 40
110  IF N<50 THEN PRINT "x=";X0 ELSE PRINT "在迭
      代 50 次后仍未得到结果"
120  END
RUN
x1, x2=-1, 0↙
  1              -.6126998        0
  2              -.5721814        0
  3              -.5677032        0
  4              -.5672056        0
  5              -.5671503        0
  6              -.5671441        0
  7              -.5671435        0
  8              -.5671433        0
x=-.5671433
```

## 13.4.4 二分法

二分法的基本方法为：设 $f(x)$ 在 $(x_1,x_2)$ 区间内为单调函数，若 $f(x_1)$ 与 $f(x_2)$ 异号，则在 $(x_1,x_2)$ 间有一实根。见图 13.9。

(1) 找 $(x_1,x_2)$ 的中点 $x_0$。

(2) 判断 $f(x_0)$ 与 $f(x_1)$ 是否同号，如同号（如图 13.9 所示），则根必在 $(x_0,x_2)$ 范围内，将 $x_0$ 作新的 $x_1$，即舍去 $(x_1,x_0)$ 区间。

(3) 若 $f(x_0)$ 与 $f(x_1)$ 异号，则舍去 $(x_0,x_2)$ 区间，将 $x_0$ 作为新的 $x_2$。

(4) 对新的 $(x_1,x_2)$，再用二分法求 $x_0=\frac{x_1+x_2}{2}$，重复以上步骤，直到 $|x_1-x_2|$ 足够小为止。

这方法的思路与弦截法有些相似，只是不作弦了，每次舍去原区间的一半。

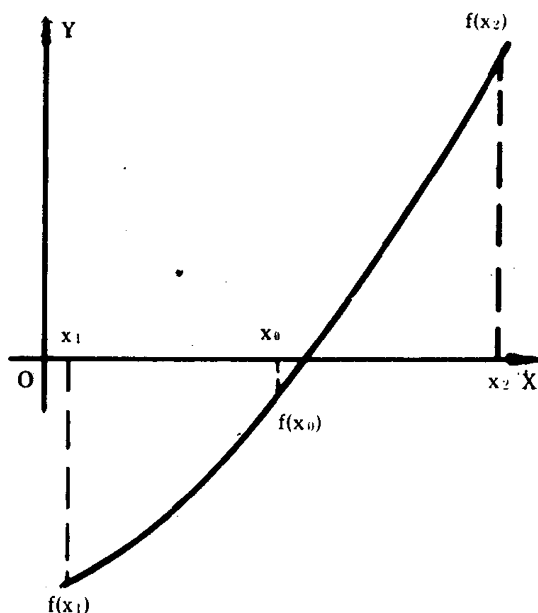


图 13.9

**【例13.9】** 用二分法求 $x+e^{+x}\doteq 0$ 的根。

有了弦截法的基础，可以直接写出程序。

```
10   DEF FNF (X) =X+EXP (X)
20   INPUT "x1, x2=", X1, X2
30   IF SGN (FNF (X1)) =SGN (FNF (X2)) GOTO 20
40   N=0
50   F1=FNF (X1): F2=FNF (X2)
60   X0= (X1+X2) /2
70   F0=FNF (X0)
80   IF SGN (FO) <>SGN (F1) THEN X2=X0: F2=F0 ELSE X1=X0: F1=F0
90   N=N+1
100  PRINT N, X1, X2
110  IF ABS (F0) >.0000001 AND N<100 GOTO 50
120  PRINT "X="; X0
130  END
RUN
x1, x2=? -1,   0↙
x=-.5671433      N=23
```

以上介绍了求一元方程的四种不同方法。如果函数变了，不是 $f(x)=x+e^x$，而是其它形式的函数，只需改变自定义的函数语句即可。

应该说明，这几种方法都是近似方法，求出的并不是根的准确值。但它能求出解析方法所不能解的方程的根。还有其它求根的方法，有兴趣的读者可以参阅有关计算方法的书。

请读者考虑：如果 $f(x)$ 在 $(x_1,x_2)$ 区间不是单调函数而只是连续函数，那么如果 $f(x_1)$ 与

$f(x_2)$ 异号，有可能不止一个根，这时应一个一个根找出来。即从 $x_1$ 开始，不断地加 $\Delta x$，直到找到一个最近的 $x_2$，$f(x_1)$ 与 $f(x_2)$ 异号，然后用上述方法找到第一个根，然后再从这个根开始（确切说从这个根加一个小的数开始），作为新的 $x_1$ 不断加 $\Delta x$，直到找到一个新的 $x_2$，$f(x_1)$ 与 $f(x_2)$ 异号，此时 $(x_1,x_2)$ 又应有一个根，再用上述方法求。

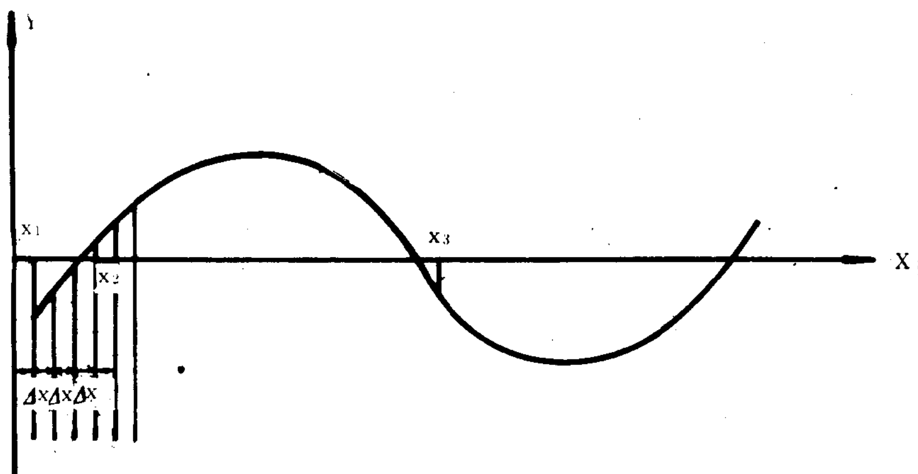



图 13. 10

图 13. 10 中表示从 $x_1$ 开始，每次加一个 $\Delta x$，直到加了三个 $\Delta x$ 后，$f(x_1)$ 与 $f(x_2)$ 异号，然后找 $(x_1,x_2)$ 区间的根。找到第一个根后，再从 $x_2$ 开始不断加 $\Delta x$，直到 $x_3$，此时 $f(x_2)$ 与 $f(x_3)$ 异号。找 $(x_2,x_3)$ 间的根，请读者自己完成编程序工作。

# §13.5 排序方法

在第六章中已介绍过用比较交换法和选择法排序。排序的方法很多，不同的排序方法效率很不一样。我们在这里再介绍三种排序方法。

## 13.5.1 起泡法

起泡法的基本思路是：将相邻两个数 A（1）和 A（2）比较，按要求将这两数排好序，再将 A（2）与 A（3）比较，…依此处理，直到将最后两个数比较并处理完毕。这时最大的数已换到最后一个数了。这是第一轮的比较和处理。每进行一轮，把剩下的数中最大的一个移到最后位置。共进行若干轮。下面结合一个具体例子来说明。

【**例 13.10**】有 6 个数，按由小到大顺序排列之。设 6 个数如图 13.11（a）所示。

第一次先将前两个数 10 和 8 比较，按题意要求小的在前，因此将 8 和 10 两数的位置对换（见图（b））。第二次将第二、三个数比较，由于 5 比 10 小，故交换（见图（c）），第三次，将第三、四个数 10 和 7 交换（见图（d）），第四次将第四、五个数 10 和 3 交换（见图（e）），第五次将第五、六个数 10 和 1 交换（见图（f））。可以看到，经过这一轮（共 5 次）比较交换，最大的数 10 已“沉底”，而最小的数 1 已“上浮”一个位置。解决了最大的数后，还要处理前面 5 个数。见图 13. 12（a）。现在要对剩下的 5 个数排序，按上述方法再进行第二轮的比较交换，见图 13. 12。第二轮中共比较 4 次。经 4 次比较后 5 个数中最大的数 8 已“沉底”，

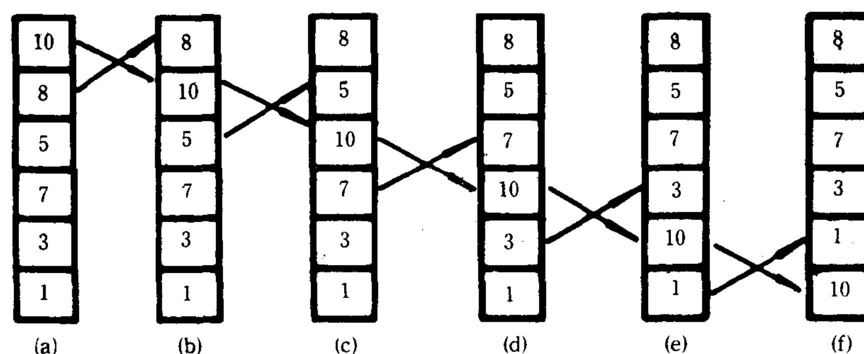



图 13.11

最小的数 1 又"上浮"一个位置。至今为止，已将 6 个数中最大两个数放到最后两个位置上了，还有 4 个数要排序。可以看出，还要进行三轮比较，每一轮把最大数"沉底"。如有有 n 个数，要进行（n—1）轮比较交换。每一轮中要比较多少次呢？可以看到：第 1 轮中比 5 次，第 2 轮比 4 次…。可以推出：第 i 轮要比（n—i）次。

可画出 N—S 图（图 13..13（a））。

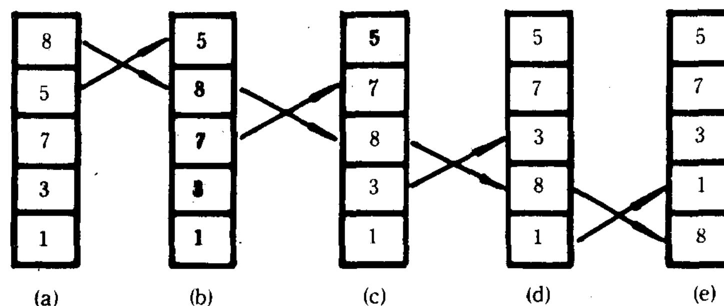
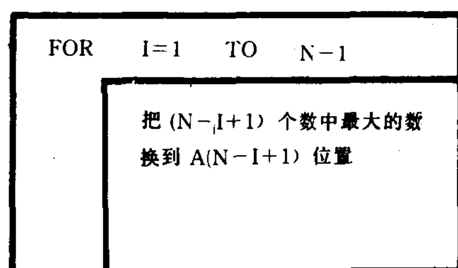
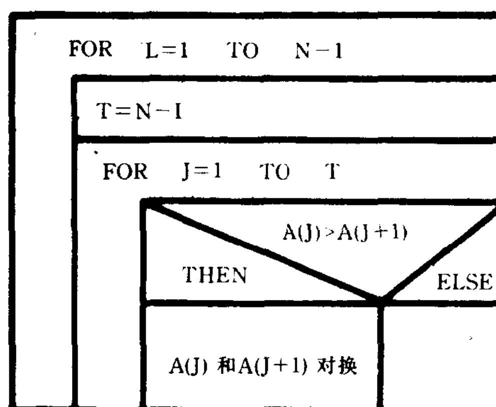



图 13.12




(a)




(b)

图 13.13

第一轮将 6—1+1=6 个数中最大的数放到 A（6）位置上。第二轮把 5 个数中最大的数放到 A

(5)，…。对图13.13 (a) 细化。得到图13.13 (b)。对 (N－I＋1) 个数进行一轮比较交换，需要比较 (N－I) 次。

据此编出程序：

```
10  INPUT "N=", N
20  DIM A   (N)
30  FOR I=1 TO N
40    INPUT A (I)
50  NEXT 1
60  FOR I=1 TO N-1
70    T=N--I
80    FOR J=1 TO T
90      IF A (J) >A (J+1) THEN C=A (J): A (J) =A (J+1): A (J+1) =C
100   NEXT J
110  NEXT I
120  PRINT : PRINT "The sorted sequence:"
130  FOR I=1 TO N
140    PRINT A (I);
150  NEXT I: PRINT
160  END
```

这种方法，大数下沉，小数逐渐“上升”，如同水中气泡上升，故称“起泡法排序”(bubble sort)。这种方法交换的次数比较多，效率并不高，尤其是当原始数列的顺序与题目要求的顺序恰恰相反时（如原始数列是递减的：10，8，6，4，2，0)，每比较一次都要交换数据。读者可自己统计一下比较多少次，交换多少次？可以在 80 行中最后加一个语句：“N=N+1”，统计交换次数。

### 13.5.2 插入法排序

基本思路是：先将第一个数作为数列中的数，然后将第二个数插入到这个数列中，使之按要求排列，以后一个一个数插入到已排序的数列中，直到所有数都插入完为止。

**【例13.11】** 用插入法对 n 个数排序。今以 7 个数为例，其排序过程如图13.14所示。

| | | | | | | |
|---|---|---|---|---|---|---|
| 初始数列 | 〔10〕 | 8 | 5 | 7 | 3 | 1 |
| i=1 | 〔8 | 10〕 | 5 | 7 | 3 | 1 |
| i=2 | 〔5 | 8 | 10〕 | 7 | 3 | 1 |
| i=3 | 〔5 | 7 | 8 | 10〕 | 3 | 1 |
| i=4 | 〔3 | 5 | 7 | 8 | 10〕 | 1 |
| i=5 | 〔1 | 3 | 5 | 7 | 8 | 10〕 |

图 13. 14

方括弧内是数列中已有的数。开始时只将第一个数 10 放在数列中。然后进行第 1 次插

入，将第二个数 8 插入，由于 8<10，故 8 应插到 10 之前，此时数列中已有两个数了。以后每次插入一个数，其插入 5 次即完成。可以用图 13.15 表示粗算法。

如何实现插入，可以将准备插入的数放在变量 $x$ 中，然后将 $x$ 与数列中最后一个数比较，如果 $x$ 大，则插入到最后，如果 $x$ 小，则 $x$ 与最后第二个数比较，如此依次从后到前逐个比较，直到 $x > A(I)$ 为止，$x$ 就插入到 $x(I)$ 之后。对图 13.15 插入操作细化，如图 13.16。

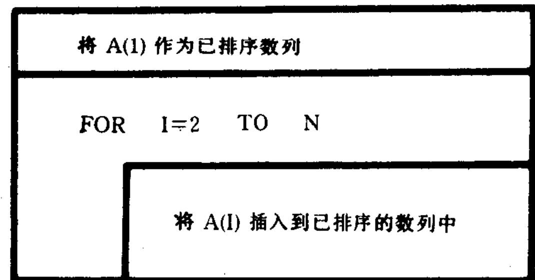



图 13.15

图 13.16 中的"J=I－1"是先求出已有数列中数的个数，要从原有数列中 [1, J] 第 1 个位置到第 J 个位置中找到一个位置将 x 插入。只要 x<A (J) 就要将 A (J) 后移一个位置 (到 A (J+1))，然后使 J=J－1，目的是使 x 与前一个数比较。当J=0 就表示已和最前面的第一个数进行过比较，不应再进行比较了。根据流程图写出程序：

```
10  INPUT "n=", N
20  DIM A (N)
30  FOR I=1 TO N: INPUT A (I): NEXT I
40  FOR I=2 TO N
50      X=A (I): J=I-1
60      WHILE J>0 AND X<A(J)
70          A (J+1) =A(J)
80          J=J-1
90      WEND
100    A(J+1)=X
110  NEXT I
120   FOR I = 1 TO N: PRINT A (I): NEXT I:
      PRINT
130  PRINT
140  END
RUN
n=5↙
?   8↙
?   4↙
?   6↙
?   0↙
?   1↙
 0    1    4    6    8
```

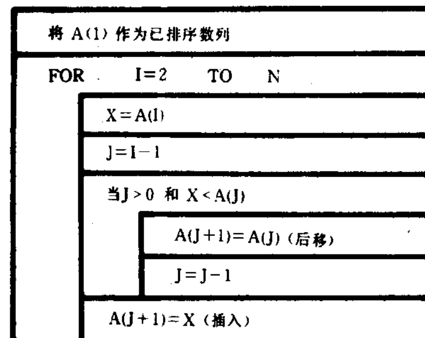



图 13.16

当 N 较小时，这种方法比较好，但当 N 很大时，比较和移动次数比较多，效率比较低。

## 13.5.3 希尔法排序

这是 D. L. Shell 提高的一种高效率的排序方法 ( *Shell's method* )，是对插入排序法的改进

和发展。插入排序算法简明，实现容易，当 N 小时效率比较高，但当 N 大时效率较低。希尔法将一个数列分为若干个子数列，每个子数列只包括为数较少的数，用插入法对子数列排序，这时的效率是高的。Shell 方法是这样分解一个数列为若干个子数列的：先定一个增量 $d_1$，从第一个数开始，把相隔为 $d_1$ 的各个数作为一个子数列，分别对每一个子数列排序。然后再取一个增量 $d_2$（$d_2<d_1$），再分子数列，再分别排序。再取 $d_3$（$d_3<d_2$），……如此反复，直到 $d_n=1$ 为止。见图 13. 17。

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 原始数列 | 9 | 7 | 8 | 3 | 1 | 0 | 6 | 5 | 4 | 2 | |
| d=4 | 9 | | | | 1 | | | | 4 | | （第一组） |
| | | 7 | | | | 0 | | | | 2 | （第二组） |
| | | | 8 | | | | 6 | | | | （第三组） |
| | | | | 3 | | | | 5 | | | （第四组） |
| 第一轮结果 | 1 | 0 | 6 | 3 | 4 | 2 | 8 | 5 | 9 | 7 | |
| d=3 | 1 | | | 3 | | | 8 | | | 7 | （第一组） |
| | | 0 | | | 4 | | | 5 | | | （第二组） |
| | | | 6 | | | 2 | | | 9 | | （第三组） |
| 第二轮结果 | 1 | 0 | 2 | 3 | 4 | 6 | 7 | 5 | 9 | 8 | |
| d=2 | 1 | | 2 | | 4 | | 7 | | 9 | | （第一组） |
| | | 0 | | 3 | | 6 | | 5 | | 8 | （第二组） |
| 第三轮结果 | 1 | 0 | 2 | 3 | 4 | 5 | 7 | 6 | 9 | 8 | |
| d=1 | 1 | 0 | 2 | 3 | 4 | 5 | 7 | 6 | 9 | 8 | （第一组） |
| 最后结果 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |

图 13. 17

开始时取 d 比较大，每组中数比较少，排序时数据移动少，效率高，而且由于相隔为 d 的数组成一组，因此小的数如果在后则以跳跃式向前交换位置，而不是一个位置一个位置地向前移，所以排序速度快。以后 d 减小，但由于各组中的数大多数已有序，因此，在实现插入时的比较和移动的次数就比较少了。Shell 排序法在整个排序过程中保持高的效率（不论 N 多大，也不论原始数据的排列状况如何）。

Shell 提出 $d_1=\text{INT}(N/2)$，即第一次增量大致为数列中数的个数的 1/2，也就是说，每个子数列中只有 2～3 个数，$d_2=\text{INT}(d_1/2)$，$d_3=\text{INT}(d_2/2)$，……。

**【例 13. 12】** 用希尔法对 N 个数排序

先画出 N－S 图（图13. 18）。可以看出：当 D=1 时，和插入排序相同。

```
10  READ N
20  DIM A(N)
30  FOR I=1 TO N: READ A (I): NEXT I
40  D=INT (N/2)
```

```
50  WHILE D>=1
60     FOR I=D+1 TO N STEP D
70       X=A (I): J=I-D
80       WHILE J>0 AND X<A(J)
90           A (J+D) =A (J): J=J-D
100          IF J<=0 THEN K=J: J=0
110        WEND
120        IF J<=0 THEN J=K
130        A (J+D) =X
140    NEXT I
150        D=INT (D/2)
160 WEND
170 REM…输出结果
180 FOR I=1 TO N: PRINT A (J);, NEXT I:
    PRINT
190 DATA 10, 9, 7, 8, 3, 1, 0, 6, 5, 4, 2
200 END
RUN
 0   1   2   3   4   5   6   7   8   9
```

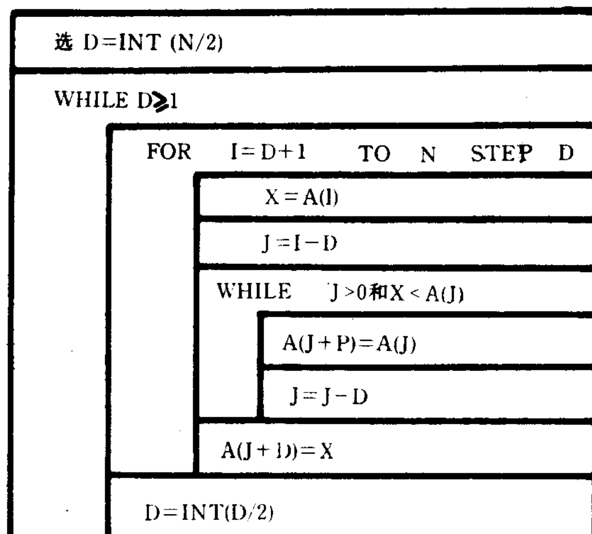



图 13. 18

在写程序时还有一个问题要处理：由于每一组中各个数的相隔位置为 D，在执行 70 行的"J=I-D"语句后 J 的值可能会小于零，如果无 100 语句，就会在 80 语句判断循环条件"J>0 AND X<A (J)"时由于 A (J) 的下标为负值而出错。我们加了 100 行，当 J<0 时，使 J=0 从而不会出现"下标为负"的错误，但是 130 语句中，A (J+D) 的 J 应是原来的 J 值而不是零，因此在 100 行中当 J<0 时先将 J 的值赋给一个临时变量 K 以保留 J 的原值，然后才使 J=0。在 120 行中再将 K 的值赋还给 J，这样执行 130 行时才正确。

我们已介绍了几种不同的排序方法。在"数据结构"课程中还要对每一种排序方法作详尽的比较分析。

排序是数据处理中一项重要的操作，例如要对全校学生按成绩排序、按职工工资排序等，不仅要对若干数值排序，还要对若干个记录按其关键字排序等，因此，掌握排序方法是十分重要的。

我们只是从算法设计的角度作介绍，以帮助读者学会思考问题和进行程序设计。

## §13.6 矩 阵 运 算

矩阵运算是数值运算中一个重要的部分，如求两个矩阵的和、差、积，求矩阵的逆，矩阵置零，矩阵转置等。有些计算机系统使用的 BASIC 有 MAT (矩阵) 语句 (MS BASIC 没有)。为使读者有一初步了解，下面列出这些矩阵语句：

| | |
|---|---|
| MAT A=ZER | 使矩阵 A 全部元素为零值 |
| MAT A=CON | 各元素均为 1 |
| MAT A=IDN | A 为单位矩阵 (主对角线元素的值为 1，其它元素为零) |
| MAT C=A+B | 矩阵加 |

MAT　C=A－B　　矩阵减

MAT　C=A * C　　矩阵乘

MAT　B=INV（A）　B矩阵为A矩阵之逆

MAT　B=TRN（A）　B矩阵为A矩阵的转置

MAT　READ A　　矩阵读

MAT　INPUT A　　矩阵输入

MAT　PRINT A　　矩阵输出

MAT　A=B　　矩阵赋值

如果所用BASIC系统无矩阵语句，则需要用循环来实现矩阵运算。

**【例13.13】** 矩阵相乘。C=A * B

两个矩阵A和B相乘的前提是：A的列数必须等于B的行数。如果A矩阵为n×m，则B矩阵应为m×p。C应为n×p。

C矩阵中各元素的值为：$C_{ik}=\sum_{j=1}^{m}A_{ij}\times B_{jk}$

设：$A=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$　　$B=\begin{pmatrix}1 & 4 & 7 & 10\\ 2 & 5 & 8 & 11\\ 3 & 6 & 9 & 12\end{pmatrix}$

则C应为2×4的矩阵。

$$C_{11}=a_{11}\times b_{11}+a_{12}\times b_{21}+a_{13}\times b_{31}$$
$$C_{12}=a_{11}\times b_{12}+a_{12}\times b_{22}+a_{13}\times b_{32}$$
$$C_{13}=a_{11}\times b_{13}+a_{12}\times b_{23}+a_{13}\times b_{33}$$
$$C_{14}=a_{11}\times b_{14}+a_{12}\times b_{24}\times a_{13}\times b_{34}$$
$$\vdots$$
$$C_{24}=a_{21}\times b_{14}+a_{22}\times b_{24}+a_{23}\times b_{34}$$

即$C_{ik}$元素的值是由A矩阵中第i行与B矩阵中第k列各元素相应的乘积之和。例如$C_{13}$的值由A矩阵第1行与B矩阵第3列各元素相应乘积之和。

据此可编写程序：

```
10  DIM A(2,3), B(3,4), C(2,4)
20  FOR I=1 TO 2: FOR J=1 TO 3, READ A (1, J): NEXT J, NEXT I
30  FOR I=1 TO 3: FOR J=1 TO 4, READ B (I, J): NEXT J, NEXT I
40  FOR I=1 TO 2
50      FOR K=1 TO 4
60          C (I, K) =0
70          FOR J=1 TO 3
80              C (I, K) =C (I, K) +A (I, J) *B (J, K)
90          NEXT J
100     NEXT K
110 NEXT I
120 PRINT "matrix c:"
130 FOR I=1 TO 2
140     FOR K=1 TO 4
```

```
150            PRINT C (I, K);
160        NEXT K
170        PRINT
180    NEXT I
190    DATA 1, 2, 3, 4, 5, 6
200    DATA 1, 4, 7, 10, 2, 5, 8, 11, 3, 6, 9, 12
210    END

RUN
matrix c:
  14     32     50     68
  32     77     122    167
```

读者可仿此用循环来实现各种矩阵运算，例如从矩阵中找出最大元素，求主对角线元素之和，求矩阵中各元素之和等。

## §13.7 计算机模拟

用计算机模仿实物系统进行测试，从测试的结果获得期望的资料，称为计算机模拟（Computer Simulation），又称仿真。计算机模拟是计算机的一项重要的应用，几乎各种实况都可以用计算机进行模拟。所谓“实况”，指的是一切真实的情况，如车间的流水作业、车辆的调度、小学生的游戏、人造卫星的飞行状况等。一个实况往往是很复杂的，牵涉到很多因素，常对它作一定的简化，用一个比较简单的模型来代替它。例如炮弹的平抛轨迹可以用抛物线作为模型表来近似模拟它（忽略空气阻力）。

根据模拟对象的不同，可以分为：(1) 确定性模拟；(2) 机率性模拟两大类。

### 13.7.1 确定性模拟

所谓“确定性模拟”(Deterministic Mode)，是指被模拟对象的模型中，各参数都是确定的值，也就是说，实况是一个完全确定的过程。例如炮弹平射，当高度、初速已知时，炮弹的运动就完全确定了。用计算机模拟这个过程，就是确定性模拟。

**【例 13.14】** 甲飞机在坐标原点（0，0）朝 $x$ 轴方向以 $V_A = 100$ 米/秒 的速度向右飞去。在同一坐标系统的 B 点（4000 米，10000 米）有乙机放出导弹，该导弹能自动校正方向正对甲飞机（图 13.19）。假设导弹飞行速度为 $V_B = 300$ 米/秒，模拟甲飞机和导弹的轨迹，求出几秒钟后导弹击中甲机，当导弹距离飞机 50 米以内，就认为“击中”。

我们用模拟的方法处理这个问题。

设甲飞机位置为 $A(x_1,y_1)$，导弹为 $B(x_2,y_2)$。开始时 $A$ 在 $O$ 点。导弹方向为 $\overline{AB}$ 方向。导弹速度 $\overline{V}_B$ 可以分解为沿 $x$ 方向的 $\overline{V}_{Bx}$ 和沿 $y$ 方向的 $\overline{V}_{By}$，由直角三角形关系可知：

$$\overline{BA}^2 = y_2{}^2 + (x_1 - x_2)^2, \qquad \overline{BA} = \sqrt{y_2{}^2 + (x_1 - x_2)^2}$$

$$V_{BX} = \frac{x_1 - x_2}{\overline{BA}} \cdot V_B, \qquad V_{By} = \frac{x_1 - x_2}{\overline{BA}} \cdot V_B = \frac{-y_2}{\overline{BA}}$$

经过 $\Delta t$ 秒后，$A$ 点位置为 $A'$：

$x_1' = V_A \times \Delta t$　　　（$V_A$ 为甲飞机速度）

$y_1' = 0$　　　（水平飞行）

B 位置变为 B'：

$$x_2' = x_2 + V_{BX} \cdot \Delta t$$

$$y_2' = y_2 + V_{By} \cdot \Delta t$$

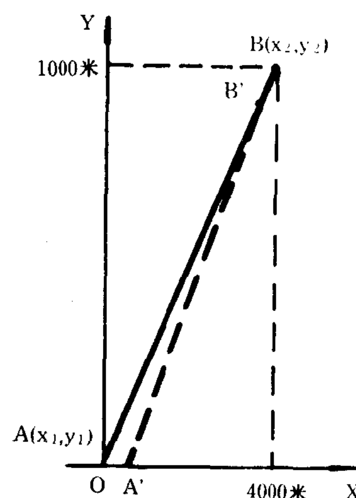



图 13.19

根据 $A$ 和 $B$ 的新位置判断是否“击中”，如“未击中”，则再计算导弹速度新的方向$\overline{B'A'}$（对准 $A'$，见图中虚线），如此每隔 $\Delta t$ 计算一次。即模拟在不同时间间隔时 $A$ 和 $B$ 的相对位置。

今以 $DX$ 代表 $\Delta x = x_1 - x_2$，$D$ 代表 $A$ 与 $B$ 的距离，$VBX$ 代表 $V_{BX}$，$VBY$ 代表 $V_{By}$，$DT$ 代表 $\Delta t$。$V_{BX}$ 为负表示方向向左，$V_{By}$ 为负表示向下。可画出 N－S 图（图 13. 20）。

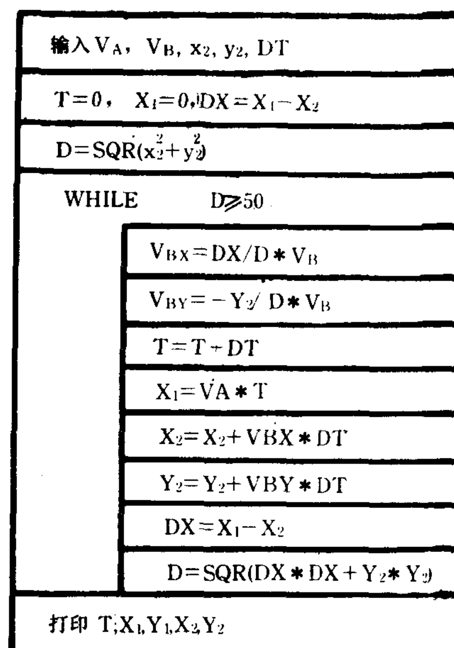



图 13.20

```
10  INPUT "va, vb="; VA, VB
20  INPUT "x2, y2=", X2, Y2
40  INPUT "dt=", DT
50  PRINT "  t", "  x1", "  y1", "  x2", "
    y2"
60  T=0
70  X1=0
80  T=0 : X1=0 : DX=X1-X2
90  D=SQR (X2*X2+Y2*Y2)
100  WHILE D>=50
130    VBX=DX/D*VB : VBY=-Y2/D*VB
140    T=T+DT
150    X1=VA*T : X2=X2+VBX*DT : Y2=
       Y2+VBY*DT
160    DX=X1-X2
170    D=SQR (DX*DX+Y2*Y2)
180  WEND
190  PRINT T, X1, Y1, X2, Y2
200  END
RUN
va, vb=  100,   300↙
x2, y2=  4000, 10000↙
dt=0.1↙
 t            x1           y1        x2          y2
35.20004      3520.004     0         3478.336    11.42461
RUN
va, vb=100,   300↙
```

```
x2, y2=4000, 10000↙
dt=0.01↙
t           x1          y1          x2          y2
35.15001    3515.011    0           3469.947    16.32819
```

当 $\Delta t=0.1$秒，得到结果为：在35.2秒时“击中”。此时 A 点在（3520.004，0），B 为（3478.336，11.42461）。当 $\Delta t=0.01$秒时，在35.15秒时“击中”，此时 A 为（3515.011，0），B 为（3469.947，16.32819）。

$\Delta t$ 愈小愈好，模拟愈真实，如果选 $\Delta t$ 过大，则可能出现“不击中”情况。例如选 $\Delta t=0.5$ 秒，每隔0.5秒才调整一次导弹的方向并计算 $AB$ 的距离，误差较大。读者不妨试一下。如果给定的 $V_A, V_B, x_2, y_2, \Delta t$ 数值不合适，可能出现“不击中”情况，读者自己可用不同数据试一下（例如 $\Delta t=1$ 秒）。

这个例子说明如何用计算机模拟在不同时间下飞机和导弹轨迹的实况。

**【例13.15】** 假设有一化学工厂，有一水池原盛盐水 100 升，每升含盐 1 千克。今以每升含盐 2 千克的盐水，每分种注入池中 5 升，并随时将池水充分拌匀。另有一个出水管将此均匀的盐水每分种放出 5 升（即池中容量始终保持 100 升），求注放开始后，池中含盐量增至 150 千克需要多少时间？

这个问题可以用两种方法求解：

（1）用一阶常微分方程求解：

$t$ —注放开始后经历的时间（单位：分）

$Q$ —池内含盐量（单位：千克），$t=0$ 时，$Q=100$

$dt$ —时间的微分增量

$dQ$ —在 $dt$ 时间内总盐量的增值

每分钟注入的盐量 $=2\times5=10$（千克）

每分钟放出的盐量 $=5\times\frac{Q}{100}=\frac{Q}{20}$（千克）

在 $dt$ 这段时间内池中增加的总盐量为：

$$dQ=(10-\frac{Q}{20})dt \tag{1}$$

这就是解本题的微分方程。

如果用数学方法解，可将（1）式化为：

$$\frac{dQ}{Q-200}=-\frac{dt}{20} \tag{2}$$

（2）式积分：

$$ln(Q-200)=-\frac{t}{20}+lnc$$

通解：

$$Q-200=ce^{-\frac{t}{20}}$$

以初始条件 $Q=100, t=0$ 代入式得：

$$100-200=ce^0=c$$

$$\therefore c=-100$$

$$\therefore Q=200-100e^{-\frac{t}{20}}$$

如 $Q = 150$，则 $t = -20ln(\frac{1}{2}) = 13.9$(分钟)

当 $t$ 趋近∞时，总盐量：

$$\lim_{t\to\infty} Q = 200 - 100 \times 0 = 200(\text{千克})$$

$$t \to \infty$$

盐量变化过程见图 13. 21。

(2) 现在我们用计算机模拟变化的过程。每过 $\Delta t$ 时间计算一次 $\Delta Q$ 和 $Q$，将 $Q$ 与要求的 150 千克相比，如未达到 150 千克，则再算 $\Delta Q$ 和 $Q$，……直到 $Q \geqslant 150$ 千克为止。

变量设置：

S—流入盐水浓度（2kg/升）

F—每分钟流量（5 升/分）

V—池水体积（100 升）

Q—池水原含盐量（100 千克）

P—要求达到的含盐总量（150 千克）

E—允许含盐量误差（取 E=0.001千克）

DT—模拟时间增量（0.1分）

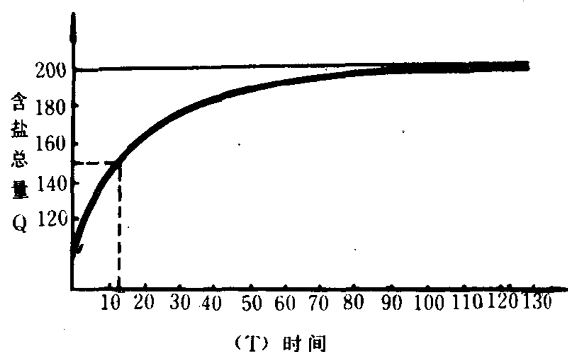



图 13.21

画出 N－S 图（图13.22）

程序如下：

```
10  REM     模拟含盐量的程序
20  INPUT "s, f, v, q, p, e, dt=", S, F, V, Q, P, E, DT
30  PRINT "模拟结果:"
40  PRINT "时间（秒）", "含盐量（公斤）"
50  S1=F*S
60  T=0 : SWITCH=0 : N=1
70  WHILE SWITCH=0
80     K=T : L=Q : S2=F*Q/V : Q1= (S1-S2) *DT
90     T=T+DT : Q=Q+Q1
100    IF INT (T+.00001) =N THEN PRINT N, Q : N=N+1
110    IF ABS (Q-P) <=E THEN PRINT T, Q, "end" : SWITCH=1
120    IF Q>P THEN T=K : Q=L : DT=DT*0.1
130 WEND
140 END
RUN
s, f, v, p, q, e, dt=2, 5, 100, 100, 150, 0.001, 0.1↙
模拟结果:
时间（秒）        含盐量（公斤）
  1              104.889
  2              109.5389
  3              113.9616
  4              118.168
```

| | |
|---|---|
| 5 | 122.1687 |
| 6 | 125.9739 |
| 7 | 129.593 |
| 8 | 133.0352 |
| 9 | 136.3091 |
| 10 | 139.423 |
| 11 | 142.3846 |
| 12 | 145.2014 |
| 13 | 147.8805 |
| 13.82802 | 149.9993 end |

若在输入时，将 e 的值改为0.0001，则输出结果的最后一行为：

13.82832　　150　　end

前面各行都不改变。

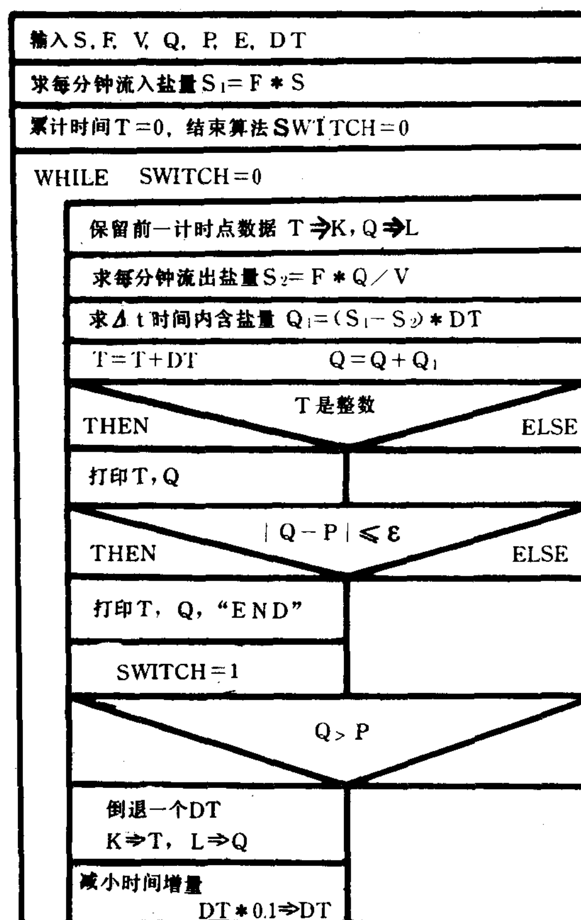



图 13.22

T 从 0 开始，开始时每隔0.1秒计算一次含盐状况(即每隔0.1秒模拟一次实况)。当 T 为整数时打印数据。当 Q=P 时（即 $|Q-P|\leqslant\varepsilon$)，打印最后的 T，P，程序作结束处理（使开关标志 SWITCH=1，从而不进入下次循环)。若 Q>P 则表示在这次增加的 DT 时间内 Q 由小于 P 变成 Q>P。(例如 T=13.8秒时，Q<P，而 T=13.9秒时，Q>P，因此 Q=P 应出现在13.8～13.9秒之间)，故应将 T 倒退一个 DT，减小 DT（例如原来 DT=0.1秒，现改为0.01秒，以便找出 Q=P 的准确时间)。K 和 L 是保留前一时间点数据的临时变量，今将 K 和 L 的值赋还给 T 和 Q，按新的 DT 重新进行模拟。如果又出现上述情况（Q>P)，再使 DT 缩小为 1/10，…，如此进行下去，可以求出一个比较准确的 T 值（参阅输出结果情况)。

请注意如何处理“当 T 为整数时打印经历时间和含盐量”。今设一变量 N，初值为 1，当 T 变化到 T=N 时就打印 T 和 Q，由于考虑到实数的存贮和运算会有些小误差(例如 T 应为 3 时实际上却为2.999999)，故先加一个小的数再取整，当此数等于 N 时，就表示 T 为整数。请分析 100 行。

## 13.7.2　机率性模拟

机率性模型 (Stochastic Mode)，指的是模型中有未确定的参数，或者说，实况的过程是不确定的，人们事先不能确切地知道实际情况将如何进行。例如，向上抛一个硬币，落到桌面上到底是正面还是反面？明年七月一日是晴天还是雨天？今天有没有交通事故？什么地方出现交通事故？明天考试某班有多少人不及格等。虽然在这些事件出现之前它们是不能确切地确定的。但人们往往希望了解它们发生的可能性究竟有多大(即机率)，以便作出必要的反应。人们可以根据过去的经验来预测明年七月一日的天气是 70%晴天，20%阴天，6%小雨，4%

大雨；可以估计出硬币落地对正反面出现的可能性将各占一半；可以根据过去的情况预测出今天有可能出现 1～2 次交通事故，发生的可能地点是郊区和繁华地段；某班有可能有 1/3 学生的数学考试不及格，等等。本书第七章例 7. 7，例 7. 8 都属于机率性模型。在用计算机模拟时要用到随机函数来模拟某个不确定的参数。例如抛 100 次硬币，可以用 100 个 0～1 之间的随机数来模拟，这些由计算机产生的随机数在 0～1 区间是均匀分布的，当随机数的个数足够多时，在 0～0.5和0.5～1 之间的个数应该大体相等。可以将随机数在0.5以下的认为代表硬币的正面，大于0.5而小于 1 的代表反面。用这个办法来求得正面和反面出现的概率。

**【例 13.16】** 有一种混凝土结构，其抗压强度随每个结构而不同，根据统计，有 5%为 210 千克/厘米²，10%为 225 千克/厘米²，25%为 240 千克/厘米²，30%为 255 千克/厘米²，25%为 270 千克/厘米²，5%为 285 千克/厘米²，见图 13. 23。结构承受的应力也是不确定的。根据统计，承受应力 165 千克/厘米²的机会为 5%，承受 180 千克/厘米²的机会为 10%，195 千克/厘米²为 20%，210 千克/厘米²为 25%，225 千克/厘米²为 25%，240 千克/厘米²为 10%，255 千克/厘米²为 5%（见图 13.24）。

| 强度或应力（千克/厘米²） | 分布机率 | |
|---|---|---|
| | 强度 | 应力 |
| 165 | | 0.05 |
| 180 | | 0.10 |
| 195 | | 0.20 |
| 210 | 0.05 | 0.25 |
| 225 | 0.10 | 0.25 |
| 240 | 0.25 | 0.10 |
| 255 | 0.30 | 0.05 |
| 270 | 0.25 | |
| 285 | 0.05 | |
| 合计 | 1.00 | 1.00 |

求该结构体的可靠度。可靠度指在上表的情况下，混凝土强度比承受的应力大的机率。

这是一个非确定性问题，因为我们不知道某一结构的具体抗压强度和具体的承受压力。只能用模拟的方法来测算其概率，计算出可靠度为多少。我们可以用计算机产生 1～100 之间（包括 1 和 100）的一个随机整数来代表强度，如果它在 1～5 之间时取抗压强度为 210 千克/厘米²(机率为$\frac{5}{100}$=5%)，若在 6～15 之间则取强度 225 千克/厘米²(机率为$\frac{10}{100}$=10%)，16～40 则取强度 240 千克/厘米²，…，96～100 时取 285 千克/厘米²。另外再产生一个 1～100 之间的随机整数代表承受的应力，若该数在 1～5 之间，则取应力为 165 千克/厘米²，在 6～15 间则取 180 千克/厘米²，16～35 则取 195 千克/厘米²，…。比较强度和应力看谁大。如此重复若干次，累计抗压强度大于所受应力的次数，由此求出了可靠度的机率。

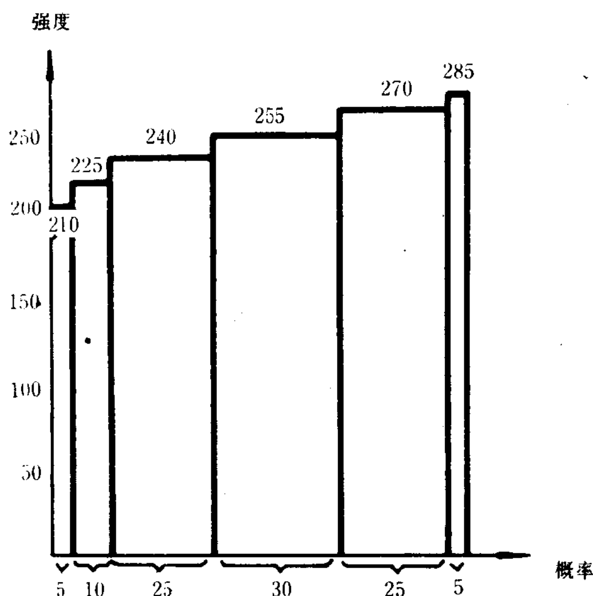



图 13.23

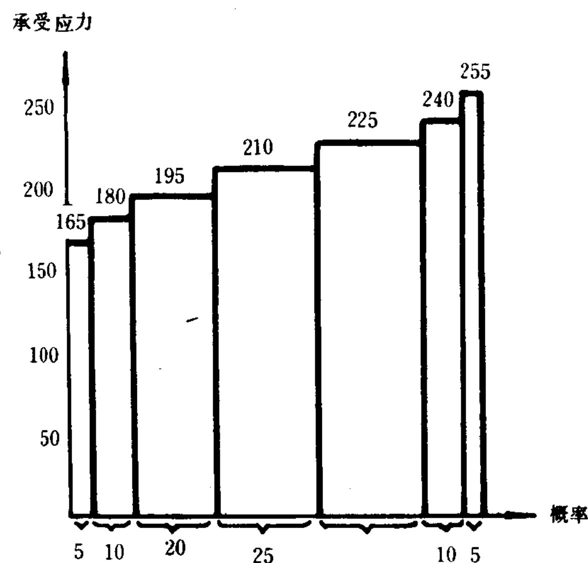



图 13.24

可以画出流程图（图13.25），并据此编出程序为：

```
5   RANDOMIZE
10  INPUT "K=", K : INPUT "L=", L
20  PRINT "模拟一个结构的可靠度"
30  PRINT "次数", "可靠度"
40  FOR I=1 TO K
50    T=0
60    FOR J=1 TO L
70      N1=INT (RND (1) * 100) +1
80       IF N1<6 THEN S1 = 210  ELSE IF N1<16
         THEN S1=225 ELSE IF N1<41 THEN S1=240
         ELSE IF N1<71 THEN S1=255 ELSE IF N1<
         96 THEN S1=270 ELSE S1=285
90      N2=INT (RND (1) * 100) +1
100     IF N2<6 THEN S1=165 ELSE IF N2<16 THEN
        S2=180 ELSE IF N2<36 THEN S2=195 ELSE
        IF N2<61 THEN S2 = 210 ELSE IF N2<86
        THEN S2=225 ELSE IF N2<96 THEN S2=240
        ELSE S2=255
110     IF S1>S2 THEN T=T+1
120    NEXT J
130    R=T/L
140    M=M+R
150    PRINT  I, R
160 NEXT I
```

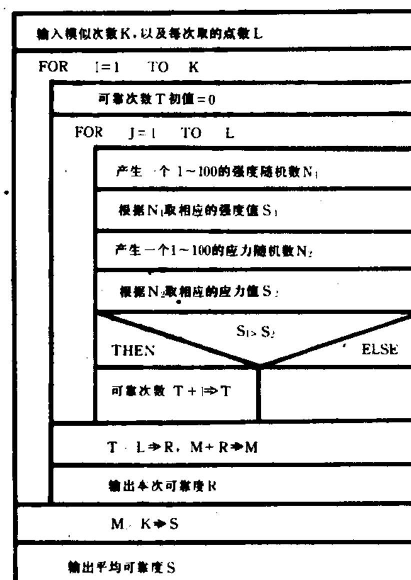



图 13.25

```
170 S=M/K
180 PRINT
190 PRINT "平均可靠度="; S
200 PRINT
210 PRINT "每一次所选的点数="; L
220 END
```

每次计算出一个“可靠度”概率，共计算 10 次，最后取其平均值。如果每次取的随机数为 L 不是 10 而是 100 或更大（如 1000），则更好些，每次循环所得到的结果也会接近些。如果 L 小，则反之。

下面是 K=10，L=10 和 K=10，L=100 以及 K=10，L=1000 时的三次运行记录。为节省篇幅，我们把三次运行结果并列，以便比较。

Random number seed (−32768 to 32767)? 3↙

K=10

| L=10 | | (L=100 时) | (L=1000) |
|---|---|---|---|
| 模拟一个结构的可靠度 | | | |
| 次数 | 可靠度 | | |
| 1 | .6 | .82 | .883 |
| 2 | .9 | .9 | .876 |
| 3 | .9 | .86 | .877 |
| 4 | .9 | .9099999 | .877 |
| 5 | .6 | .9 | .878 |
| 6 | .9 | .87 | .886 |
| 7 | 1 | .88 | .876 |
| 8 | .6 | .9 | .884 |
| 9 | .9 | .93 | .876 |
| 10 | .9 | .86 | .866 |
| 平均可靠度= | .82 | .883 | .8778999 |
| 每一次所选的点数=10 | | 100 | 1000 |

模拟是在处理实际问题时十分有用的一种方法。

## §13.8 用高斯消元法解一元联立方程组

一元 n 阶线性联立方程组的一般形式为：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 & (1) \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 & (2) \\ \qquad\qquad\qquad\qquad\qquad\vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n & (n) \end{cases}$$

在代数学中一般用消元法来解方程组。即：先将第一行乘以一个常数再与其它行相加，以消去其它各行的 $x_1$ 那一项（使 $a_{21}$，$a_{31}$，$\cdots a_{n1}$ 为 0）。然后再以新的第二行乘以一个常数并与第 3 行到第 n 行相加，以消去第 3 行到第 n 行上 $x_2$ 的那一项（使 $x_2$ 的系数为 0）。…最后再以

新的第（n－1）行乘一个常数并与第 n 行相加，以消去第 n 行上的 $x_{n-1}$ 项。最后得到一个如下形式的三角方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ \qquad a_{22}'x_2 + a_{23}x_3 + \cdots + a_{2n}'x_n = b_2' \\ \qquad\qquad a_{33}'x_3 + \cdots + a_{3n}'x_n' = b_3' \\ \qquad\qquad\qquad \vdots \\ \qquad\qquad\qquad a_{nn}'x_n = b_n' \end{cases}$$

此过程可用图13.26表示。

从此方程组最后一个方程式可以直接求出 $x_n = \frac{b_{n'}}{a_{nn'}}$。然后逐步“回代”，求出 $x_{n-1}$，$x_{n-2}\cdots$，x1。

还要考虑一个问题：如果在上面过程中，$a_{ii}$ 为零，则在消元过程中会出现使第 i 行乘以常数 $\frac{a_{ki}}{a_{ii}}$ 而出现无穷大，溢出。例如本来为了消去第 2 行的 $x_1$ 项，要进行的是：(1) 式 $\times\frac{a_{21}}{a_{11}}-$ (2) 式，若 $a_{11}=0$，则发生溢出错误。必须保证 $a_{ii}\neq0$。若发生 $a_{ii}$ 为零，可从第 i＋1 行到第 n 行中找到一个第 m 行，其 $a_{mi}\neq0$，将此第 m 行与第 i 行对调。如果找不到，则方程无解或无定解。可用图 13.27 表示。

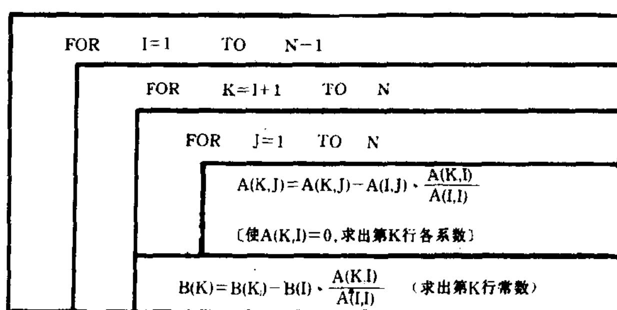



图 13. 26

根据上面介绍的方法，利用图 13. 27 所示的 N—S 图，得到上面列出三角方程组。再使该三角方程组中各 $A_{ii}$ 的值为 1，以得到以下三角方程组：

$$\begin{cases} x_1 + a_{12}''x_2 + a_{13}''x_3 + \cdots + a_{1n}''x_n = b_1 \\ \qquad x_2 + a_{23}''x_3 + \cdots + a_{2n}''x_n = b_2 \\ \qquad\qquad \vdots \\ \qquad\qquad x_{n-1} + a_{(n-1)n}''x_n = b_{n-1} \\ \qquad\qquad\qquad x_n = b_n'' \end{cases}$$

这样就可得到 $x_n=b''_n$。然后回代。求出 $x_{n-1}$，$\cdots x_1$ 的值。画出这部分流程图（图 13. 28）。

请读者自己根据图 13. 27 和图13.28写出解一元阶线性方程组的程序。为清晰起见，最好用子程序，一个子程序完成一个功能。

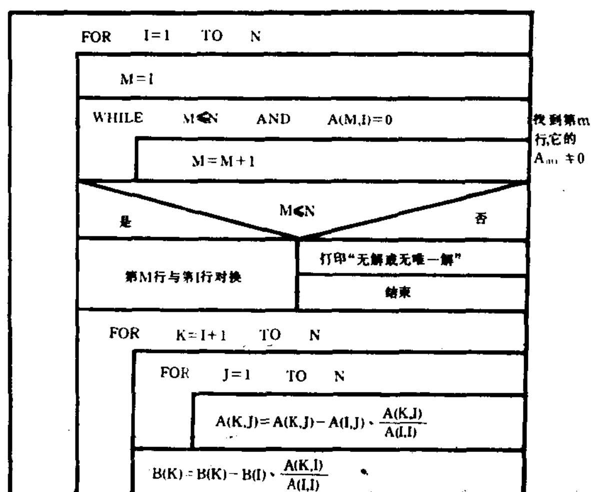



图 13.27

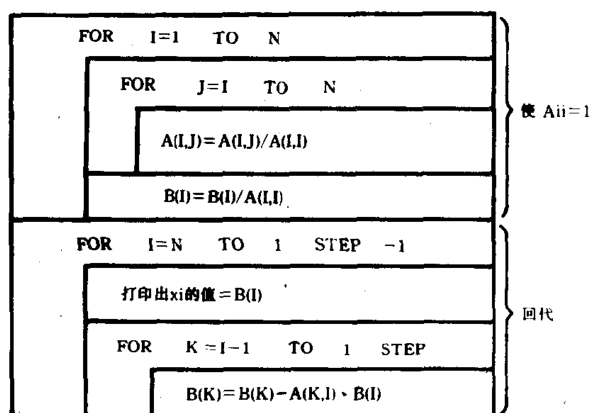



图 13.28

## 习　　题

13.1　用一元票换一分、二分、五分的硬币，问有哪几种方案（各多少数）。

13.2　找出 1～1000 之间的全部“同构数”。所谓“同构数”是指这样一个数：它出现在它的平方数的右端。例如 5 的平方是 25，5 是 25 中右端的数，5 就是一个同构数。25 的平方是 625，因此 25 也是同构数。

13.3　有 5 个人坐一排，问第 5 人年龄，回答“是第 4 人的一半多 2 岁”，问第 4 人年龄，回答“是第 3 人的一半多 2 岁”。…每一人都比前一人的一半多 2 岁，最后问第一个人，回答说“100 岁”，问每个人的年龄？

13.4　分别用矩形法、梯形法和辛普生法求下列函数的定积分。

(1)　$\int_0^1 x \cdot e^x dx$　　　(2)　$\int_{-1}^{1} (\sin x+\cos x)^2\, dx$

(3)　$\int_0^{0.5} e^{x/2} \cdot e^{-x/2} dx$

要求用自定义函数。并上机运行程序，比较三种方法的特点（对同一函数的值作比较，分析其近似程度）。

13. 5　用牛顿迭代法求方程在 0 附近的根

$$f(x) = x^5 - 3x^2 + 2x + 1$$

13. 6　用起泡法、插入法和希尔法，对 10 个学生记录排序，每个记录包括学号、成绩和班级。要求按分数由高到分打印出各学生的学号、成绩和班级。用同一组数据。在程序中统计比较次数和交换次数或移动次数。比较三种排序方法。上机运行程序。

13. 7　有一个有序数列，今要求插入 n 个数，插入后使数列仍保持有序。

13. 8　有一个矩阵 (n×m)，各元素值不相同，要求找出并打印其中最大和最小值元素所在位置，并将它的二者对换，然后，打印出新的矩阵。

13. 9　打印螺旋方阵。n×n 方阵中由连续的自然数 ($1\sim n^2$) 按以下规律排列（各数转圈排列）。要求输入 n，要求输出方阵。

| | | | | |
|---|---|---|---|---|
| 1 | 16 | 15 | 14 | 13 |
| 2 | 17 | 24 | 23 | 12 |
| 3 | 18 | 25 | 22 | 11 |
| 4 | 19 | 20 | 21 | 10 |
| 5 | 6 | 7 | 8 | 9 |

13. 10　用筛选法求 2～100 间的全部素数。所谓筛选法是先将 2 以后各数中 2 的倍数去掉，再将 3 以后的 3 的倍数去掉，再将 5 以后的 5 的倍数去掉，…直到将$\sqrt{100}$的整数部分 (10) 以后的 10 的倍数去掉为止，剩下的就是素数。

13. 11　模拟玩骰子的游戏，每个骰子为一个正六面体，每一面分别刻有 1、2、3、4、5、6 个点。假设有 4 个骰子，希望求出出现四个“六点”的机会是多少。

13. 12　有一球以 10 米/秒的速度向仰角 45°方向抛去，球着地后仍保持向原抛出方向跳去，但速度每跳一次比前次减少 10%。模拟此球最前面三个周期的运动，打印出其轨迹。模拟的时间间隔为0.1秒。（见图 13. 29。）

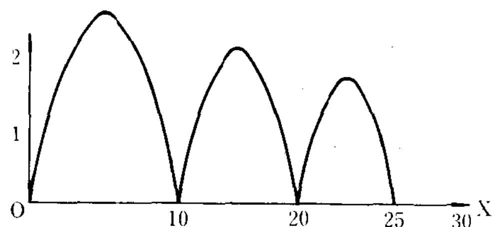



图 13. 29

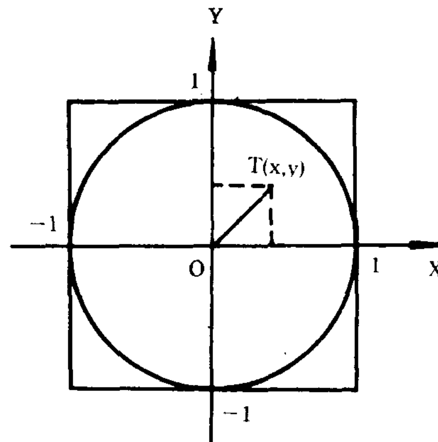



图 13. 30

13. 13　甲飞机以 500 米/秒的速度直线飞行，乙飞机由其正后方 800 米处向甲射击，炮弹以 1000 米/秒的初速射出，如炮弹每0. 1秒减速 1%，问炮弹射出后多久可以击中甲飞机。

13. 10　用蒙特卡洛法 (Monte—carlo Methed) 求 π 的近似值。这是用随机数进行统计的方法。见图 13. 30。

方法是：产生若干对随机数 x 和 y，它们的值在 (0, 1) 之间，且为正值，每一对 x，y 对应一个点 T (x, y)，T 与原点 O 的距离为$\sqrt{x^2+y^2}$，若 T<1，表示该点在半径 r=1 的圆内。统计 M 组 x，y 中有几组落在圆内。如果有 N 组落在圆内，则第一象限中的 1/4 圆面积与小正

方形面积之比为：N/M。小正方形面积为 1×1=1。整个大正方形面积为 4，因此圆面积应为 $S=4\times\frac{N}{M}$。π 的近似值$=\frac{S}{r^2}=4\times\frac{N}{M}$。

用计算机产生 M=100，1000，2000 个随机数，求 π 近似值。

13.15　求函数 $f(x)=x^2-4x+5$　在（1，3）之间的最小值。

13.16　13 个人围坐一圈，每人按顺序有一个标号 1#，2#…13#，现在，从第一个人开始数起，每数到 5 时，这个人就从圈里出来，再继续数 1，2，…，5，再数到第 5 位的这个人也从圈里出来，凡是从圈里出来的位置，下次数的时候就跳过去不再数。不断继续数下去，直到全部 13 个人从圈内出来为止。写一个程序，使能打印出从圈内出来的人的顺序。如：第一个出来的是 5#，计算机打印 5，第二个出来的是 10#，计算机打印 10，第三个出来的是 2#，计算机打印 2，…。

13.17　根据 § 13.8 介绍的方法和流程图，写出程序解以下的联立方程：

$$\begin{cases} 3x + y + 3z = 10 \\ -2x + 2y + 4z = 9 \\ 4x - 3y - 2z = -6.5 \end{cases}$$

13.18　解联立方程：

$$\begin{cases} x_1 + 3x_2 + 2x_3 - x_4 = 8 \\ 2x_1 - x_3 + 2x_4 = 7 \\ -x_2 + 2x_3 - 3x_4 = -8 \\ 3x_1 + \frac{1}{2}x_2 + 2x_4 = 12 \end{cases}$$

# 第十四章 True BASIC 简介

## §14.1 引 言

以“小型、通用、会话、易学”为特点的 BASIC 语言，自 1964 年问世以来，在世界上得到广泛的应用，这大大出乎 BASIC 的创始人、美国计算机语言学家 John G. Kemeny 和 Thomas E. Kurtz 两位教授的预料。当然，这是值得高兴的事。但是，它的成功也带来了一些问题，众多单位和企业、公司应用不同的计算机系统，它们都配置不同的 BASIC 版本，使 BASIC 具有严重的“方言性”，它们扩展的一些特性过多地依赖于硬件，从而违背了“通用性”的原则。同时，随着时间的推移，它的一些弱点也日益显露出来了，因此遭到一些批评，以至很多人怀疑它是否有生命力。

还是这两位 BASIC 创始人自己，他们用大量的事实回顾了 BASIC 语言的诞生和发展，承认并指出目前广为流传的 BASIC 版本的一些缺点，在充分吸收 FORTRAN 和 PASCAL 等语言优点的基础上，于 1985 年提出了一种完全结构化的 BASIC 最新版本—True BASIC。他们认为，True BASIC 可以与 PASCAL 相媲美，并相信它将开辟 BASIC 发展的新纪元。True BASIC 严格按照美国国家标准，并可以实现结构化、模块化程序设计。

本章将对 True BASIC 的结构化、模块化特征及绘图功能作一简要介绍。

## §14.2 True BASIC 的结构化语句

True BASIC 允许使用行号和无行号两种形式，但提倡不用行号，这样做会使得使用 GOTO 语句的欲望逐渐减弱，最终争取完全废除行号，影响结构化的 GOTO 语句也将随之消失。

**【例 14.1】** 计算 π 值。采用的方法是基于几何学中的简单事实：半径为 1 的圆面积是 π。我们可以通过计算正多边形的面积来逼近圆的面积，从圆的内接正方形开始。设变量 a 为其面积，因此，a 的初值应是 2。如果把圆心和各顶点相连，那末每个三角形的高 h 应等于 $1/\sqrt{2}$。若每次将多边形的边数加倍，其面积也就越来越接近圆的面积了。通过计算，可以得到：新的 a=a/h；新的 h=SQR（（1+h）/2）。程序和运行结果如下：

```
!    ----Find pi
     let a=2
     let h=1/sqr (2)
     for k=1 to 10
         let a=a/h
         let h=sqr ( (1+h) /2)
```

```
        print a
    next k
    end
```

---

```
Ok. run
  2.82843
  3.06147
  3.12145
  3.13655
  3.14033
  3.14128
  3.14251
  3.14157
  3.14159
  3.14159
Ok.
```

可见，程序可以不用行号；注释行用惊叹号表示；每行写一个语句；赋值语句要用语句定义符用 LET；程序可以用小写字母书写；循环体采用缩进格式。从这个例子中，主结构是 FOR－NEXT 循环，每循环一次，对应的边数就增加一倍。九次迭代以后就可精确到小数点后五位。

结构化程序设计是以顺序、选择和循环三种基本结构为基础的，结构化程度的提高，很大程度上决定于选择和循环两个结构。下面我们分析 True BASIC 中，这两个结构的特征。

### 14.2.1 True BASIC 的选择结构

True BASIC 中，保留了 BASIC 中原有的二分支选择结构：

IF〈条件〉THEN〈语句〉

IF〈条件〉THEN〈语句〉ELSE〈语句〉

此外，还增加了 IF 型结构语句和 CASE 型结构语句。

(1) IF 型结构语句

它的一般形式是格式 I 和格式 II 所示：

```
IF〈条件〉THEN
  语句块 1
ELSE
  语句块 2
END IF
```

格式 I

```
IF〈条件 1〉THEN
  语句块 1
ELSE IF〈条件 2〉THEN
  语句块 2
ELSE IF〈条件 3〉THEN
  语句块 3
ELSE
  语句块 4
END IF
```

格式 II

它们都是由 IF 语句开始，以 END IF 语句结束，中间部分分别以 ELSE 或 ELSE IF 分隔成两个或多个语句块，每个语句块可以由多行语句组成。从形式上与 FORTRAN 语言中的块 IF 一样，同样具有良好的结构化特征。
它们的结构化流程图如图 14. 1 和图 14. 2 所示：

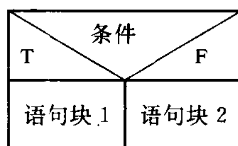


图 14.1

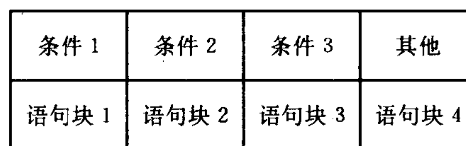


图 14.2

其中，〈条件〉是逻辑表达式，ELSE 及其下面的语句块可以没有。对于格式 Ⅱ，一个 IF 语句中，可以有多个 ELSE、IF 子句。

**【例14. 2】** 求二次方程的根。其结构流程图如图 14. 3 所示。

程序如下：

```
!    ----Quadratic equation
     print "Coefficients:";
     inprt a, b, c
     let d=b * b—4 * a * c
     if d>0 then
          let s=sqr (d)
          print "The roots are:"
          print (—b+s) / (2 * a)
          print (—b—s) / (2 * a)
     elseif d=0 then
          print "The root is (two same
                 roots):"
          print —b/ (2 * a)
     else
          print "No real roots."
     end if
     end
```

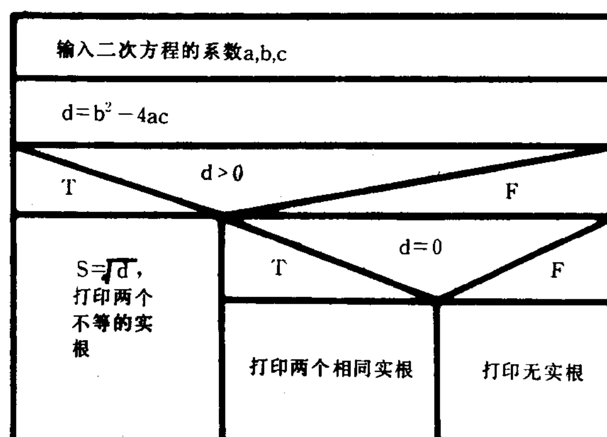


图 14.3

---

```
Ok. run
Coefficients:? 1, —2, —3↙
The roots are:
  3
—1
Ok. run
Coefficients:? 1, —6, 9↙
The root is (two same roots):
  3
Ok. run
Coefficients:? 2, 4, 5↙
```

No real roots.

Ok.

三种情况分别在 IF、ELSE IF 和 ELSE 中表示，末尾用一个 ENDIF 表示块 IF 的结束。测试是按自然顺序，由于对任何给定的 d 值都只有一条通路，所以就没有必要用 GOTO 语句了。

（2） CASE 型结构语句，一般形式是：

```
SELECT CASE〈表达式〉
CASE〈值域 1〉
    语句块 1
CASE〈值域 2〉
    语句块 2
    ⋮
CASE ELSE
    语句块 n
END SELECT
```

其结构化流程图如图 14. 4 所示，它的执行过程是，先计算表达式的值，其值在〈值域 1〉，则执行语句块 1，否则在〈值域 2〉，则执行语句块 2…，否则，执行语句块 n。

| 表达式 | | | | |
|---|---|---|---|---|
| 值域 1 | 值域 2 | 值域 3 | …… | 其他 |
| 语句块 1 | 语句块 2 | 语句块 3 | | 语句块 n |

图 14. 4

其中，〈表达式〉是字符串表达式或数值表达式，不能是逻辑表达式。〈值域 1〉，〈值域 2〉，… 〈值域 n－1〉的数据类型要跟表达式值的类型一致，且可有如下三种形式：

① 〈常量 1〉 （表示某个值，如 100）

② 〈常量 1〉 to 〈常量 2〉 （表示值域，如 “A” to “E” 或 10 to 100）

③ is〈关系运算符〉〈常量〉 （表示值域，如 is>100）

与 IF 型结构语句中的 ELSE 一样，SELECT CASE 结构中的 CASE ELSE 及其下面的语句块可以没有。

**【例 14. 3】** 统计学生的成绩，分数段规定是 100≥p≥90 为优秀；90<p≤80 为良好；80>p≥70 为中等；70>p≥60 为及格；p<60 为不及格，并求学生的总数。

这里取分数 p>100 或 p<0 为程序的结束标志。学生总数为 n，程序如下：

```
!    ············Statistics of student's marks
     let supper, upper, mid, pass, nopass=0
     input prompt "p=": p
     do while (p<=100) and (p>=0)
     select case p
             case 90 to 100
                 let super=super+1
```

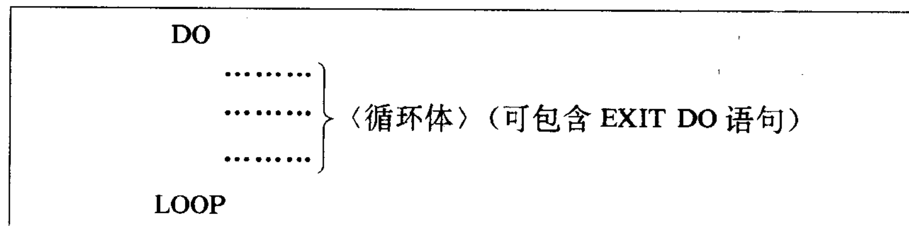
```
            case is >=80
                let upper=upper+1
            case is >=70
                let mid=mid+1
            case is >=60
                let pass=pass+1
            case else
                let nopass=nopass+1
        end select
        input prompt "p=": p
    loop
    let n=super+upper+mid+pass+nopass
    print "n=": n
    print "super=": super, "upper=": upper,
    print "mid=": mid, "pass=": pass, "nopass=": nopass
    end
Ok, run
p=88↙
p=96↙
p=68↙
p=75↙
p=78↙
p=84↙
p=56↙
p=78↙
p=-1↙
n=   8
super=   1      upper=   2      mid=3      pass=1      nopass=1
Ok.
```

## 14.2.2 True BASIC 的循环结构

True BASIC 除保留原有的 FOR－NEXT 循环结构外，还增加了 DO 循环结构：

（1） 最简单的 DO 循环语句和 EXIT DO 语句

其格式是：



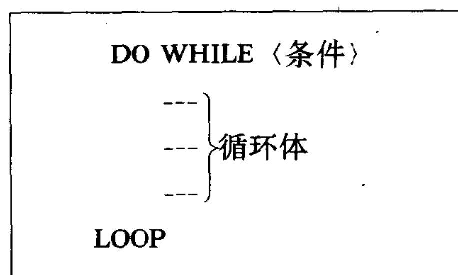
```
DO
    ………
    ………  〉（循环体）（可包含 EXIT DO 语句）
    ………
LOOP
```

其中，EXIT DO 语句能控制流程转到循环体的外部，它常常是嵌在 IF 语句中使用，当满足某

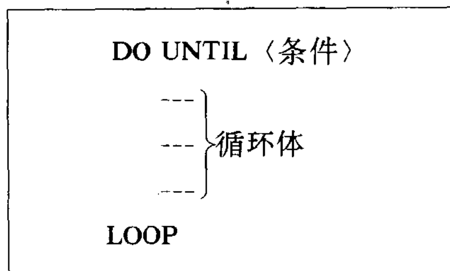
条件时，流程由循环体内转向体外。若无此语句，循环为无限循环。

(2)　WHILE 型和 UNTIL 型 DO 循环语句

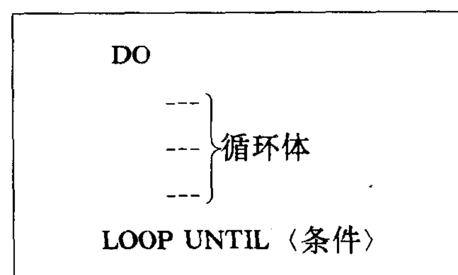
True BASIC 提供了多种灵活的循环结构形式，在简单 DO 循环语句的 DO 后面或 LOOP 后面可以用 WHILE〈条件〉或 UNTIL〈条件〉来结束循环。其格式有下列 6 种：



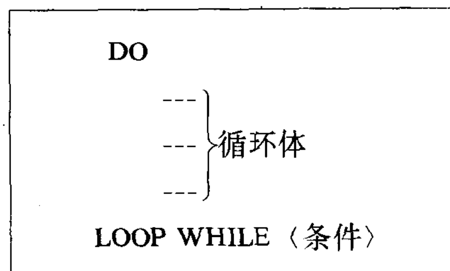

格式 1



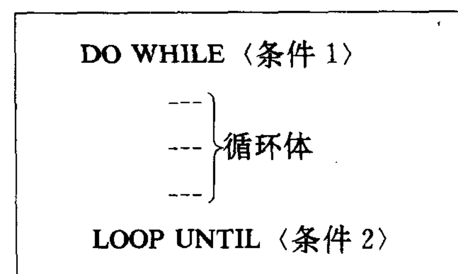

格式 2



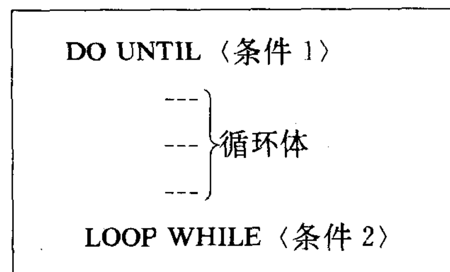

格式 3




格式 4




格式 5




格式 6

格式 1 和格式 2 为先判断条件，后执行循环体；而格式 3 和格式 4 是先执行循环体后判断条件；格式 5 和格式 6 为以上两类的结合。它们的流程图如图 14. 5 (a) 到 (f) 所示。

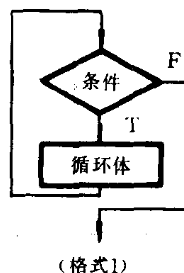
也可以用 N—S 图表示。假设“条件”以 P 表示，“条件 1”以 $P_1$ 表示，“条件 2”以 $P_2$ 表示…。格式 1 用图 14. 6 (a) 表示。格式 2 实际上是格式 1 的变形，它可以表示为图 14. 6 (b)。其中 $\overline{P}$ 表示为 P 的反条件。例如：“DO WHILE x＞0”，可以改写为：“DO UNTIL x＜＝0”，“x＜＝0”是“x＞0”的反条件。“DO WHILE x＞0…”表示：当＞0 时执行循环体，“DO UNTIL x＜＝0…”表示：若 x＜＝0 就不执行循环体。显然二者的作用相同。格式 2 也可以直接表示为图 14. 6 (b’) 形式。格式 3 用图 14. 6 (c) 表示。格式 4 其实是格式 3 的变形，可以表示为图 14. 6 (d)。$\overline{P}$ 是对 P 的取反。例如，“DO……LOOP UNTIL x＜＝0”相当于“DO……LOOP WHILE x＞0”。格式 4 可直接用图 14. 6 (d’) 表示。格式 5 可以用 (e) 图表示。格式 6 可以用图 (f) 表示，也可以画成 (f’) 形式。

要注意的是：

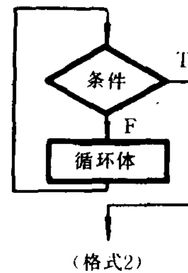
(1) 格式 1 和 2 可能一次也不执行循环体，而格式 3 和 4 至少执行循环体一次。

(2)〈条件〉可以是任意的逻辑表达式。

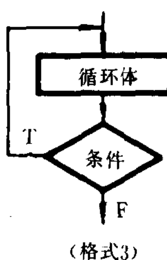
(3) WHILE 是当条件为真时，重复执行循环体；而 UNTIL 是当条件为真时，停止执行循环体。因此，要把 WHILE 改成 UNTIL 时或 UNTIL 改为 WHILE 时，条件要变反。



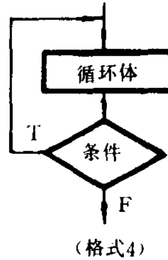
(格式1)



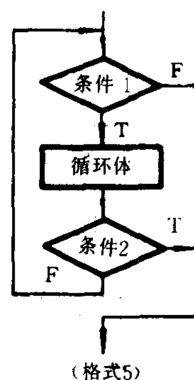
(格式2)



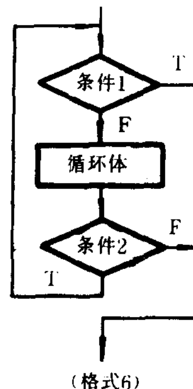
(格式3)



(格式4)



(格式5)



(格式6)

图 14.5

例 14.4 同时展示了 EXIT DO 语句和 WHILE 型 DO 语句的用法，程序很简单，目的只是了解一下这两个语句的应用。

【例14.4】 计算两数乘积（在 DO 循环中应用 WHILE〈条件〉和 EXIT DO 语句来决定程序是否退出循环）。程序如下：

```
!     ----Calculates products
      do
           input prompt "Input a, b:": a, b
           if a=0 then exit do
           print "a*b="; a*b
           print "more";
           input answer $
      loop while answer $ = "yes"
      end
```

---

```
Ok. run
Input a, b: 8, 5↙
a*b=40
more? yes↙
Input a, b: 6, 4↙
a*b=24
more? no↙
Ok.
Ok. run
Input a, b: 3, 4↙
a*b=12
more? yes↙
Input a, b: 0, 8
Ok.
```

输入两个乘数，计算机打印乘积，当回答“yes”时可以继续输入乘数。否则，结束程序的运行。程序中有意放入一个条件语句，当输入的第一个乘数为零时，通过 EXIT DO 语句可以退出循环。

其中，LOOP 中的条件，我们可以用下面语句来代替：

LOOP UNTIL ANSWER $ = "NO"

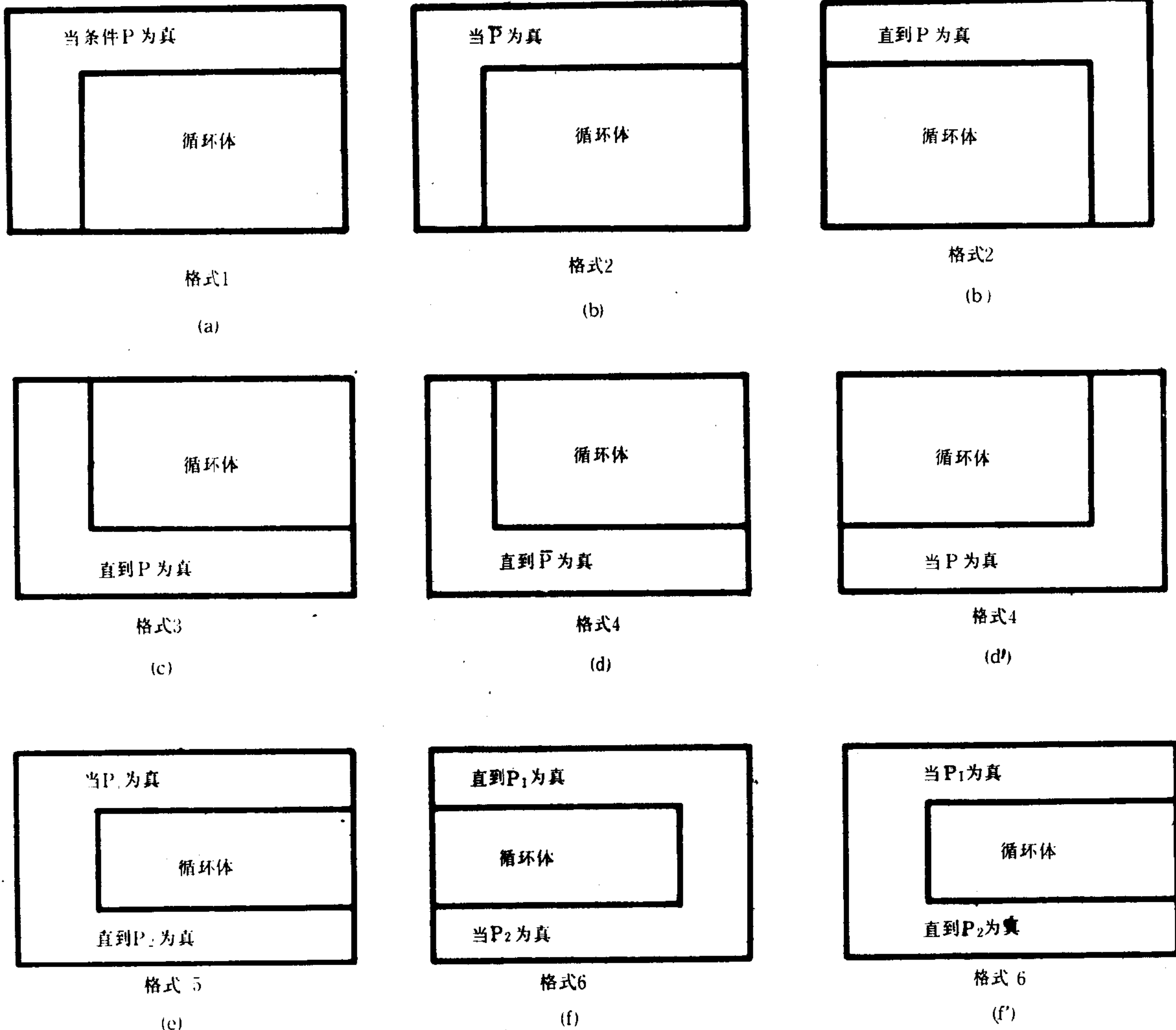



图 14.6

## §14.3 True BASIC 的程序的模块化

True BASIC 允许用户定义函数、子程序和绘图功能，用它们实现模块的功能。并且可以把它们组成"库文件"，以便调用，这就为实现程序的结构化、模块化提供了有力的工具。

### 14.3.1 函数的定义和调用

True BASIC 保留了 BASIC 中的自定义函数，除了可以使用单行定义形式外，还增加了多行定义形式：

```
DEF〈函数名〉(〈参数 1〉,〈参数 2〉,…)
    ---
    ---  语句块
    ---
    LET〈函数名〉=〈表达式〉
    ---
    ---  语句块
    ---
END DEF
```

函数可以有一个或多个参数，但调用后只由函数名带回一个函数值。位置在调用程序 END 语句之前的函数称为内部函数，如果函数的位置在 END 语句之外，则称为外部函数。内部函数的调用和标准函数一样，而外部函数调用时，首先要在程序开头用如下语句说明：

```
DECLEARE DEF〈函数名 1〉,〈函数名 2〉,…
```

其中一次可以说明多个函数名，用逗号隔开。然后和标准函数一样调用。

## 14.3.2 子程序的定义和调用

True BASIC 中，子程序都有自己的名字，并且用 CALL 语句来调用，子程序由 SUB 开头，由 END SUB 结束，很容易识别，它是一段独立的程序块，可以放在程序中的任何地方。根据子程序的位置在调用程序的 END 语句之前或之后，分别称为内部子程序和外部子程序。

```
SUB〈子程序名〉[(〈参数 1〉,〈参数 2〉,…)]
    ---
    ---  语句块
    ---
END SUB
```

子程序的调用形式是：

```
CALL〈子程序名〉[(〈表达式 1〉,〈表达式 2〉,…)]
```

这里，CALL 语句中列出的表达式的值的数据类型必须与上面 SUB 语句中列出的参数相匹配。不管是内部子程序还是外部子程序，调用的形式是相同的。

## 14.3.3 库文件

库文件不包含主程序，而是一些外部函数和外部子程序的集合，以便供不同程序共享。True BASIC 除了允许用户以库文件的形式存放自己定义的外部函数和外部子程序外，还提供了四个包含数学函数的库，一个包含图画的库和一个包含使用菜单子程序的库。

(1) 库文件的建立

首先在几个无主程序的外部函数或外部子程序前加一条 EXTERNAL 语句，然后用 SAVE 命令将它们存入磁盘，形式是：

| SAVE〈库名〉 |
|---|

〈库名〉遵循 DOS 中有关文件名的规定。

(2)　库文件的使用

当程序要调用库中的子程序或函数时，应在调用之前使用一条 LIBRARY 语句，并指出要检索的库文件名。形式是：

| LIBRARY "〈库名 1〉"，"〈库名 2〉"，… |
|---|

这样就可用通常方式调用库 1 和库 2 中的函数和子程序了。在实现比较复杂的算法时，利用库文件的优越性是明显的。

# § 14.4　True　BASIC 的绘图

True　BASIC 提供了很好的绘图功能，它的绘图语句简单、直观，同时，可以定义子程序形式的图画程序块。

## 14.4.1　图形窗口坐标

True　BASIC 把屏幕看作是一个平面直角坐标，图形窗口就是平面坐标系中的一个矩形区间，图就画在此矩形区间之内。所以，在使用作图语句前，必须先用设置窗口语句：

| SET WINDOW〈Xmin〉，〈Xmax〉，〈Ymin〉，〈Ymax〉 |
|---|

所建立的窗口如图14.7所示。

用户设置窗口的同时，也就设置了窗口的纵横比。坐标与象素点的换算由 True　BASIC 自动进行。如果需要，可以把几个不同窗口的图形映射到同一屏幕上，组成一幅新的图形。

True　BASIC　用 CLEAR 语句清除当前图形窗口。

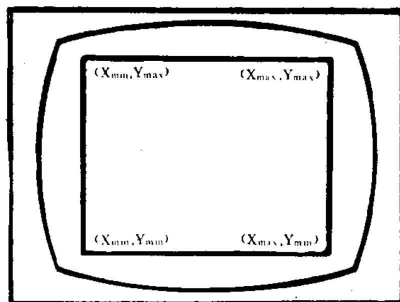



图 14.7

### 14.4.2 作图

（1） 画点：TrueBASIC 使用 PLOT POINTS 语句画点，形式是：

| PLOT POINTS：$X_1$，$Y_1$〔；$X_2$，$Y_2$；…〕 |
|---|

其中，$X_i$，$Y_i$ 是要画的点的坐标。如果仅画一个点，则语句可以简写为：

| PLOT X，Y |
|---|

（2） 画线：

True BASIC 提供的画线语句的形式是：

| PLOT LINES：$X_1$，$Y_1$；$X_2$，$Y_2$；… |
|---|

或

| PLOT $X_1$，$Y_1$；$X_2$，$Y_2$；… |
|---|

其中分号表示从前一个坐标点向后一个坐标点画一条直线。如 PLOT 5，5；20，20 表示由坐标点（5，5）向坐标点（20，20）画一直线。而

PLOT 0，0；1，0；1，1；0，0

表示画一个直角三角形，两条直角边的长度是 1。

当 PLOT 语句以分号结尾时，表示其最后一个坐标点，将向下一个 PLOT 语句中的第一个坐标点画一直线，如上面这条语句跟下面两条语句的作用是一样的。

```
PLOT 0，0；1，0；
PLOT 1，1；0，0
```

其实在应用 PLOT 语句时，当坐标点后有分号时，称定点有向输出，即向下一个坐标点画线；而坐标点后无分号时，称定点无向输出。

（3） 画矩形

True BASIC 提供一个专用画矩形的语句，形式是：

| BOX LINES Xmin，Xmax，Ymin，Ymax |
|---|

矩形框中的坐标是以窗口坐标为基准的。例如窗口坐标设定为：

```
SET WINDOW －14，14，－10，10
```

时，若 BOX 语句是下面的形式：

```
BOX LINES －5，5，－5，5
```

则程序画出的是在坐标正中的边长为 10 的正方形。

（4） 画圆和画椭圆

BOX 语句还可以用来画圆和椭圆，形式是：

| BOX CIRCLE Xmin，Xmax，Ymin，Ymax |
|---|

或

| BOX ELLIPSE Xmin，Xmax，Ymin，Ymax |
|---|

这两种语句通用，当“盒子”是正方形时，画出的是圆；当“盒子”是长方形时，画出的是椭圆。

### 14.4.3 着色

（1） 前景颜色和背景颜色

True BASIC 用 SET COLOR 语句设置前景颜色，形式是：

SET COLOR "〈色名〉"

或

SET COLOR〈色号〉

True BASIC 用 SET BACK 语句设置背景颜色（底色），形式是：

SET BACK "〈色名〉"（或〈色号〉）

或

SET BACKGROUND COLOR "〈色名〉"（或〈色号〉）

IBM—PC 机上使用的前景色和背景色分别如表 14.1 和表 14.2 所示。

表 14. 1

| 色号 | 色名 |
|---|---|
| 0 | 同背景色 |
| 1 | green（绿） |
| 2 | red（红） |
| 3 | yellow（黄） |
| 4 | cyan（青蓝） |
| 5 | magenta（洋红） |
| 6 | white（白） |

表 14. 2

| 色号 | 色名 |
|---|---|
| 0 | black |
| 1 | blue |
| 2 | green |
| 3 | cyan |
| 4 | red |
| 5 | magenta |
| 6 | yellow |
| 7 | white |

（2） FLOOD 语句

FLOOD 语句用来着色，其形式是：

FOLLD X，Y

其功能是将点（X，Y）的颜色扩散到具有相同颜色的邻近象原点上。它可用来对用当前色画的封闭图形着色。

（3） PLOT AREA 语句

语句的形式是：

PLOT AREA：$X_1$，$Y_1$；$X_2$，$Y_2$；…

其功能是先从第一点（$X_1$，$Y_1$）到第二点（$X_2$，$Y_2$）画线，再从第二点到第三点（$X_3$，$Y_3$）画线，以此类推，最后画一条最后点到第一点的线，把区域封闭起来，再用前景色填满整个区域。

（4） 用 BOX AREA 语句代替 BOX LINES 语句，可以画出一个用当前颜色填满的矩形块。

**【例14.5】** 在同一屏幕上画出正弦曲线和常用对数曲线，以便观察它们的交点在哪

儿。

程序用 SET WINDOW 语句指定我们希望看到的平面范围，我们把它置成 X 从 0 到 10，Y 从 -1 到 1，然后清除屏幕，画上 X 轴和 Y 轴，X 轴和 Y 轴的颜色是绿色，正弦曲线的颜色是用黄色，而常用对数曲线的颜色是用红色。此程序在黑白机（具有图形适配器的）上运行时，两根轴和两根曲线都显示同一颜色。程序如下：

```
!    ----Intersection of two curves
     set window 0, 10, -1, 1
     clear
     set color "green"
     plot 0, 0; 10, 0
     plot 0, -1; 0, 1
     set color "yellow"
     for x=0 to 10 step 0.2
         plot x, sin (x);
     next x
     plot
     set color "red"
     for x=1 to 10 step 0.2
         plot x, log10 (x);
     next x
     end
```

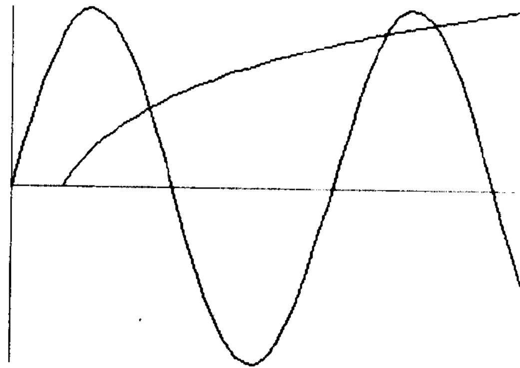


图 14. 8

图14.8是运行结果。

### 14.4.4 图形中的正文设置

图形中的正文设置可由一条 PLOT TEXT 语句来完成，它的形式是：

| PLOT TEXT, AT X, Y:〈字符串表达式〉 |
|---|

其作用是把字符串绘制在图象坐标（X，Y）的右上方。它的输出形式跟 PRINT 语句相同，也可以使用 USING $ 函数。

**【例 14.6】** 画直方图。程序运行结果图14.9所示。

```
!    ----Draw a histogram
     set window 0, 6, 0, 150
     for x=1 to 5
         read value
         box area x-.4, x+.4, 0, value
         plot text, at x-.4, value+10: using $ ("###.#", value)
     next x
     data 18.8, 40.8, 95.4, 65.6, 25.6
     end
```

### 14.4.5 动画

True BASIC 为实现动画功能，引入了保存、删除和显示图形三个语句。

（1） 把“盒子”内的图形（包括前景和后景）存入一个字符串变量之中，句子的形式是：

| BOX KEEP Xmin, Xmax, Ymin, Ymax IN 〈字符串变量〉 |
|---|

（2） 删除“盒子”的语句形式为：

| BOX CLEAR Xmin, Xmax, Ymin, Ymax |
|---|

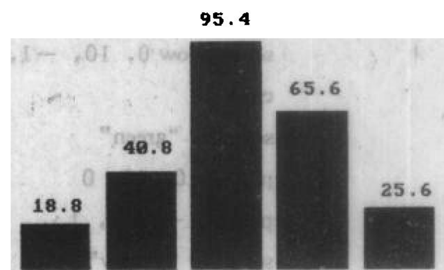



图 14.9

（3） 再现“盒子”的语句形式为：

| BOX SHOW 〈字符串变量〉 AT Xmin, Ymin |
|---|

它给出的 Xmin 和 Ymin 为盒子再现时左下角的坐标位置。

**【例14.7】** 用 True BASIC 的动画功能，画一个绕地球旋转的“人造卫星”。程序如下：

```
!      ----A moving satlate
       set window -14, 14, -10, 10
       set color "red"
       box circle-0.2, 0.2, -0.2, 0.2
       flood 0, 0
       box keep -0.2, 0.2, -0.2, 0.2 in ball $
       box clear -0.2, 0.2, -0.2, 0.2
!
       set color "blue"
       box circle -6, 6, -6, 6
       flood 0, 0
       for a=0 to 200 * pi step pi/200
               let x=8 * cos (a)
               let y=8 * sin (a)
               box show ball $ at x, y
               box clear (x), (x+0.5), (y), (y+0.5)
       next a
       end
```

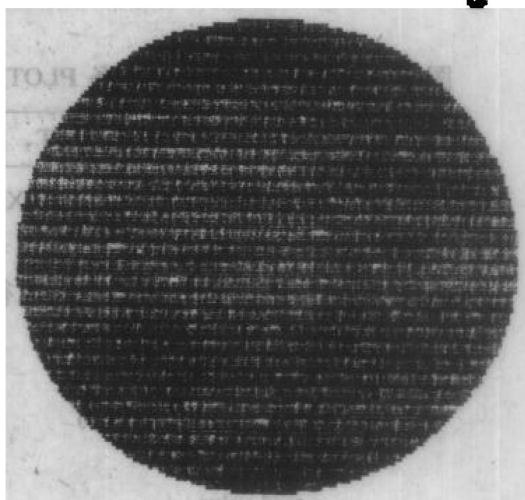


图 14.10

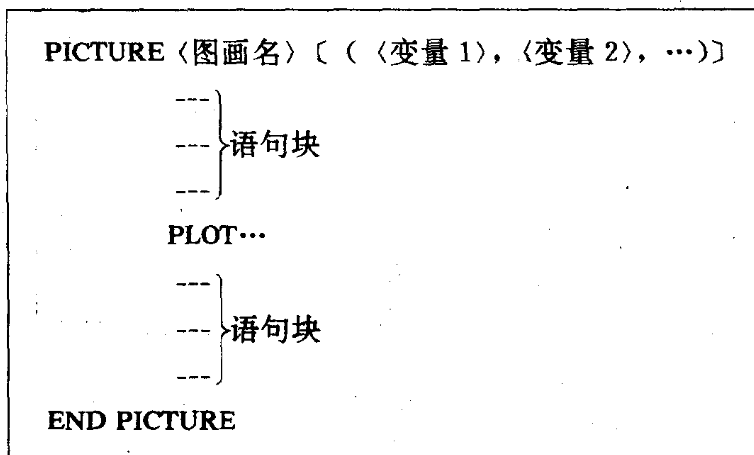
程序在某一时刻的运行结果如图 14.10 所示。

## 14.4.6 图画功能

True BASIC引入了PICTURE子程序，它与科学计算和数据处理中的子程序一样，也具有独立性，可以被其他程序调用，并且根据需要在调用它时可作移位、改变宽度、放大缩小、旋转和倾斜等五种变换。PICTURE子程序也可分内部的（定义在主程序的END之前）和外部的（定义在END之后），也可放在库文件中。

（1） 图画的定义与调用

图画的定义形式是：




图画的调用由DRAW语句后跟图画名来实现的，既可以嵌套调用，也可以递归调用。DRAW语句的形式有：

① DRAW〈图画名〉

② DRAW〈图画名〉WITH〈变换〉

③ DRAW〈图画名〉WITH〈变换〉 * 〈变换〉 * …

（2）图画的变换

True BASIC提供表14.3所示的五种变换，供图画调用语句DRAW使用。上面形式③的DRAW语句使用〈变换1〉 * 〈变换2〉是一种复合变换，可以同时进行两种或两种以上的变换。

表14.3

| 变换名 | 含义 |
|---|---|
| Shift (a, b) | 移动 (x, y) 到 (x+a, y+b) |
| Scale (a, b) | 改变 (x, y) 为 (x*a, y*b) |
| Scale (a) | 改变 (x, y) 为 (x*a, y*a) |
| Rpotate (a) | 围绕原点按逆时针方向旋转a弧度 |
| Shear (a) | 将垂直线向右倾斜a弧度 |

注意：只能对用PLOT语句画的图形作变换，而对用BOX语句画的图形无效。

**【例14.8】** 对矩形作旋转和放大变换。程序如下：

```
!      ----Make a "square" picture.
!               Rotate and Scale it.
!
      picture square (size)
            plot —size, size; size, size;
            plot  size, —size; —size, —size;
            plot —size, size
      end picture
!
      set window —14, 14, —14, 14
      for k=1 to 10
            draw square (1) with roate (k * pi/40) * scale (k)
      next k
      end
```

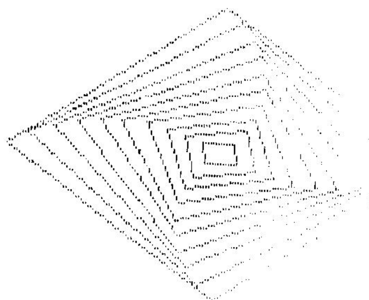


图 14.11

图14.11是程序运行结果。

**【例14.9】** 画一面五星红旗

五星红旗有五颗星，我们可以把画一个五角星编成图画子程序，然后在主程序中调用并变换它，使得它在不同的位置上按规定的比例和角度画出五颗星，星的颜色是黄色的，旗子是红色的。程序如下：

```
!     ----A flag with five stars
      set window —5, 21, —14, 6
      set back 1
      set color "red"
      box area —2.5, 10.5, —7, 3
      set color "yellow"
      draw star with scale (1, 1) * rotate (pi/2)
      option angle degrees
      for m=0 to 3
        let fi=—45+m * 30
        draw star with rotate(fi) * shift (1.5 * cos (fi), 1.5 * sin (fi)) * scale (0.4, 0.45)
      mext m
      end
      picture star
          let bgn=pi/2
          let pio=4 * pi/5
          for k=0 to 4
             let x=0.8 * cos (bng+k * pio)
             let y=0.8 * sin (bng+k * pio)
             plot x, y;
          next k
          plot 0.8 * cos (bng), 0.8 * sin (bng)
          flood 0, 0
```

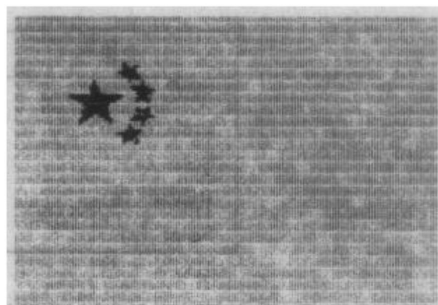


图 14.12

```
    for k=0 to 4
      flood 0.4 * cos (bng+k * pio), 0.4 * sin (bng+k * pio)
    next k
end picture
```

程序从上而下：设置图形窗口坐标；设置背景色为蓝色；设置前景色为红色；画红的旗子；设置前景色为黄色；画大五角星；选择角度单位为度；画四颗围绕的星；程序结束。END 语句下面是一个画五角星的子程序。图14.12是程序运行结果。

## §14.5 True BASIC 中的矩阵运算语句（MAT 语句）

True BASIC 还提供了一系列处理整个数组的 MAT 语句，只要用单一语句，就可以实现矩阵的读、写、赋值及加、减、乘的运算。

（1） 矩阵读

| MAT READ 〈数组 1〉，〈数组 2〉，… |
|---|

（2） 矩阵打印

| MAT PRINT 〈数组 1〉，〈数组 2〉，… |
|---|

或

| MAT PRINT 〈数组 1〉；〈数组 2〉；… |
|---|

其中逗号和分号的作用与 PRINT 语句相同。

（3） 矩阵赋值

| MAT 〈数组〉 = 〈常数〉 |
|---|

或

| MAT 〈数组 1〉 = 〈数组 2〉 |
|---|

（4） 矩阵键盘输入

| MAT INPUT 〈数组 1〉，〈数组 2〉，… |
|---|

（5） 矩阵运算

| MAT 〈数组 3〉 = 〈数组 1〉 + 〈数组 2〉 |
|---|
| MAT 〈数组 3〉 = 〈数组 1〉 − 〈数组 2〉 |
| MAT 〈数组 3〉 = 〈数组 1〉 * 〈数组 2〉 |

**【例14.10】** 假定一个班级最多可以有 50 个学生，每个学生最多有 10 个分数，计算每个学生的“加权平均分”。

要求解此问题，让我们先设几个数组，分数存放在数组“grades”中，权重放在“weights”中，算出的平均分放在“average”中，人名放在“name$”中。

所有的信息都放 DATA 语句中。先读入具体的学生数和每个学生的分数个数，然后用一个简单的 MAT 语句就可读入学生的全部分数。如：

MAT READ grades (s, g)

将读入 s 行 g 列的数组。

下面再用一个 MAT 语句就可算出所有平均值，最后打印出人名和平均分的表。

当学生人数很多时，通常把数据放在一个文件中，也可用 MAT READ 语句读文件中的数据。

程序和运行结果如下：

```
!     ----Averaging grades
!     Allow up to 50 students and 10 grades per student
      dim   grades (50, 10), weights (10), average (50), name$ (50)

      read s, g
      mat read grades (s, g), weights(g),name$(s)
      mat average=grades * weights
      for i=1 to s
      print name $(i),average(i)
      next i

      data 6, 4
      data 87, 78, 82, 86
      data 87, 99, 82, 100
      data 73, 82, 67, 85
      data 59, 62, 73, 81
      data 88, 88, 88, 88
      data 77, 61, 59, 93
      data 0.2, 0.2, 0.2, 0.4
      data Wang, Li, Zhang, Fun, Ling, Qing
      end
```

---

```
Ok. run
Wang                    83.8
Li                      93.6
Zhang                   78.4
Fun                     71.2
Ling                    88.
Qing                    76.6
Ok.
```

## § 14.6 怎样使用 True BASIC

### 14.6.1 进入 True BASIC

True BASIC 要在 MS—DOS2. 0 或更高的版本支持下工作。

当开机之后，显示器显示 A>，把 True BASIC 盘放入 A 驱动器，并键入 HELLO：

A>HELLO↙

稍等片刻，在屏幕底部显示如下：

True BASIC here

Version 1.0

Copyright (c) 1985 by True BASIC, Inc.

Published by Addison—Wesley Publishing Company, Inc.

OK

“OK”是 True BASIC 的提示符，在这儿可以键入 True BASIC 命令。

### 14.6.2 屏幕上的窗口

在进入 True BASIC 状态之后，屏幕分为上下两个区域，上面的区域叫编辑窗口(editing window)，供用户编辑自己的 True BASIC 程序用。下面的区域叫背景窗口(history window)，供用户输入 True BASIC 命令，或显示程序运行结果。由于它能给出人机对话期间的痕迹，故取名背景窗口。如图14.13所示。

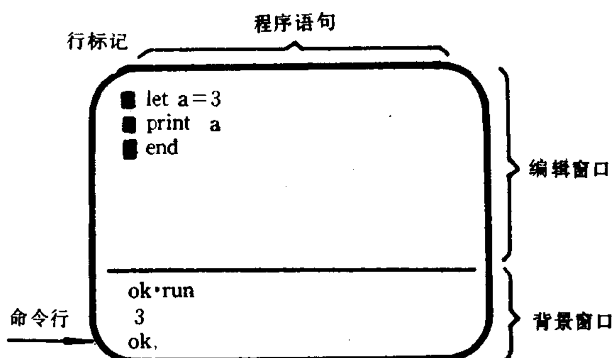



图 14.13

① 窗口转换。

两个窗口，一个可以输入 True BASIC 命令，一个可以编辑用户程序，光标可以在两个窗口上来回转换。按 $F_1$ 键，光标移至编辑窗口；按 $F_2$ 键，光标移至背景窗口，而且仍回到它离开时的位置。

② 窗口大小。

编辑窗口系统既定值是 17 行，背景窗口是 7 行。中间横线占 1 行。利用如下命令，可以重新定义编辑窗口大小：

OK. split n　　　　$(0 \leqslant n \leqslant 24)$

表示编辑窗口为 n 行。

### 14.6.3 退出 True BASIC

退出 True BASIC 可键入如下命令：

OK. bye

显示：A>—

表示机器已退出了 True BASIC 回到 DOS 状态，此时，可以把 True BASIC 盘从驱动器中取出。关机结束。

### 14.6.4 编辑命令

True BASIC 的编辑功能是全屏幕状态下进行的。表 14.4 是在 IBM—PC 机上进行编辑时，光标移动键一览表；表 14.5 是常用编辑键一览表。

表 14.4

| | | | |
|---|---|---|---|
| ← | 左移一字符位置 | PgUp | 向上翻页 |
| → | 右移一字符位置 | PgDn | 向下翻页 |
| ↑ | 上移一行 | Tab | 左移一字 |
| ↓ | 下移一行 | Shift—Tab | 右移一字 |
| Home | 移至窗口顶部 | Ctrl—← | 移至行首 |
| End | 移至窗口底部 | Ctrl—→ | 移至行尾 |

表 14.5

| | | | |
|---|---|---|---|
| 退格键 | 删除左边一字符 | F4 | 使行标志闪动，为删除、复制、移动、压缩或编辑命令作准备，也可解除当前行的标记。 |
| Del | 删除当前字符 | | |
| Ctrl—Home | 删除左边一个字（词） | F5 | 在编辑窗口可以将有标记行模块复制到光标所在行的紧后面。在背景窗口，光标在命令行时可以复制上一行命令 |
| Ctrl—PgUp | 删除当前一个字（词） | | |
| ESC | 删除当前行中光标左边的全部字符 | | |
| Ctrl—End | 删除当前行中光标右边全部字符 | F6 | 移动带标记的行模块到光标所在位置 |
| Del | 光标在行标志处时，删除当前行 | >或< | 使标记行模块右移或左移一格 |
| F7 | 重新存入被删除的当前行 | 空格键 | 当光标位于行标志处时，将当前行连接到上一行 |
| ← | 在光标所在处插入一空行 | Ins | 插入或替换字符 |
| F10 | Help 帮助 | | |

## 14.6.5 True BASIC 的文件处理和程序控制命令

表14.6列出了 True BASIC 常用的文件处理和程序控制命令。

表 14.6

| 命　令 | 参数 | 功　能 |
|---|---|---|
| New | | 清除内存，光标移到编辑窗口起点。 |
| Run（或F9） | | 解释运行内存中的程序（也叫当前文件）。 |
| Save | 文件名 | 把当前文件登目，并存入文件名。TRU 名下。如果已存在同名文件，显示出错信息，内存不变。 |
| Unsave | 文件名 | 删除盘上指定文件。文件名可用 DOS 规定的格式。 |

续表

| 命　令 | 参数 | 功　能 |
| --- | --- | --- |
| Old | 文件名 | 清除内存，调指定文件进入内存，并显示在屏幕上。 |
| Replace | 文件名 | 删除软盘上指定文件，并将当前文件存入该名下。如果盘有写保护，结果可以删除，但不能写入，显示出错。内存不变。 |
| List | [参数] | 缺省打印当前文件，内存不变。参数：$L_1$—$L_2$ 或模块名，打印指定行或模块。 |
| File | [参数] | 显示 True BASIC 盘上文件目录。参数与 DOS 的 DIR 命令类似 |
| Help 或 F10 | [参数] | 此时显示提供的解释清单，按任一键返回原状态。如参数为 topics，显示所有所能提供的 Help 参数清单。 |
| Compile | | 把当前文件编译成编译代码。存盘后，文件名后缀为 TRC |

## 习　题

14.1　请比较 True BASIC 与 MS　BASIC 的特点。True BASIC 在哪些方面作了重大改进？

14.2　True BASIC 为什么允许不用行号？而且提倡不用行号？怎样才能在程序中不用 GOTO 语句？True BASIC 提供了哪些功能以使程序可以避免使用 GOTO 语句？

14.3　用 True BASIC 编写程序，解一元二次方程：$Ax^2+Bx+C=0$。A，B，C 由键盘输入。

14.4　用 True BASIC 编写程序求 3～100 之间的全部素数。

14.5　用 True BASIC 编写程序求 $f(x)=x-e^x=0$ 的根。分别用迭代法、弦截法、牛顿迭代法、二分法解。用函数或子程序分别实现以上几种运算。

14.6　用 True BASIC 编程序画 5 个同心圆，它们的颜色不断变换。

14.7　用 True BASIC 画一个你自己设计的彩色图案（用 PICTURE 子程序）。

# 附录 I　常用字符与 ASCII 代码对照表

| ASCII 值 | 字符 | 控制字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 | ASCII 值 | 字符 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | (null) | NUL | 032 | (space) | 064 | @ | 096 | ` | 128 | Ç | 160 | á | 192 | └ | 224 | α |
| 001 | ☺ | SOH | 033 | ! | 065 | A | 097 | a | 129 | ü | 161 | í | 193 | ┴ | 225 | β |
| 002 | ● | STX | 034 | " | 066 | B | 098 | b | 130 | é | 162 | ó | 194 | ┬ | 226 | Γ |
| 003 | ♥ | ETX | 035 | # | 067 | C | 099 | c | 131 | â | 163 | ú | 195 | ├ | 227 | π |
| 004 | ♦ | EOT | 036 | $ | 068 | D | 100 | d | 132 | ä | 164 | ñ | 196 | ─ | 228 | Σ |
| 005 | ♣ | ENQ | 037 | % | 069 | E | 101 | e | 133 | à | 165 | Ñ | 197 | ┼ | 229 | σ |
| 006 | ♠ | ACK | 038 | & | 070 | F | 102 | f | 134 | å | 166 | ª | 198 | ╞ | 230 | μ |
| 007 | (beep) | BEL | 039 | ' | 071 | G | 103 | g | 135 | ç | 167 | º | 199 | ╟ | 231 | τ |
| 008 | ◘ | BS | 040 | ( | 072 | H | 104 | h | 136 | ê | 168 | ¿ | 200 | ╚ | 232 | Φ |
| 009 | (tab) | HT | 041 | ) | 073 | I | 105 | i | 137 | ë | 169 | ⌐ | 201 | ╔ | 233 | Θ |
| 010 | (line feed) | LF | 042 | * | 074 | J | 106 | j | 138 | è | 170 | ¬ | 202 | ╩ | 234 | Ω |
| 011 | (home) | VT | 043 | + | 075 | K | 107 | k | 139 | ï | 171 | ½ | 203 | ╦ | 235 | δ |
| 012 | (form feed) | FF | 044 | , | 076 | L | 108 | l | 140 | î | 172 | ¼ | 204 | ╠ | 236 | ∞ |
| 013 | (carriage return) | CR | 045 | - | 077 | M | 109 | m | 141 | ì | 173 | ¡ | 205 | ═ | 237 | Ø |
| 014 | ♫ | SO | 046 | . | 078 | N | 110 | n | 142 | Ä | 174 | « | 206 | ╬ | 238 | ε |
| 015 | ☼ | SI | 047 | / | 079 | O | 111 | o | 143 | Å | 175 | » | 207 | ╧ | 239 | ∩ |
| 016 | ► | DLE | 048 | 0 | 080 | P | 112 | p | 144 | É | 176 | ░ | 208 | ╨ | 240 | ≡ |
| 017 | ◄ | DC1 | 049 | 1 | 081 | Q | 113 | q | 145 | æ | 177 | ▒ | 209 | ╤ | 241 | ± |
| 018 | ↕ | DC2 | 050 | 2 | 082 | R | 114 | r | 146 | Æ | 178 | ▓ | 210 | ╥ | 242 | ≥ |
| 019 | ‼ | DC3 | 051 | 3 | 083 | S | 115 | s | 147 | ô | 179 | │ | 211 | ╙ | 243 | ≤ |
| 020 | ¶ | DC4 | 052 | 4 | 084 | T | 116 | t | 148 | ö | 180 | ┤ | 212 | ╘ | 244 | ⌠ |
| 021 | § | NAK | 053 | 5 | 085 | U | 117 | u | 149 | ò | 181 | ╡ | 213 | ╒ | 245 | ⌡ |
| 022 | ▬ | SYN | 054 | 6 | 086 | V | 118 | v | 150 | û | 182 | ╢ | 214 | ╓ | 246 | ÷ |
| 023 | ↨ | ETB | 055 | 7 | 087 | W | 119 | w | 151 | ù | 183 | ╖ | 215 | ╫ | 247 | ≈ |
| 024 | ↑ | CAN | 056 | 8 | 088 | X | 120 | x | 152 | ÿ | 184 | ╕ | 216 | ╪ | 248 | ° |
| 025 | ↓ | EM | 057 | 9 | 089 | Y | 121 | y | 153 | Ö | 185 | ╣ | 217 | ┘ | 249 | • |
| 026 | → | SUB | 058 | : | 090 | Z | 122 | z | 154 | Ü | 186 | ║ | 218 | ┌ | 250 | · |
| 027 | ← | ESC | 059 | ; | 091 | [ | 123 | { | 155 | ¢ | 187 | ╗ | 219 | █ | 251 | √ |
| 028 | (cursor right) | FS | 060 | < | 092 | \ | 124 | ¦ | 156 | £ | 188 | ╝ | 220 | ▄ | 252 | ⁿ |
| 029 | cursor left) | GS | 061 | = | 093 | ] | 125 | } | 157 | ¥ | 189 | ╜ | 221 | ▌ | 253 | ² |
| 030 | (cursor up) | RS | 062 | > | 094 | ∧ | 126 | ~ | 158 | Pt | 190 | ╛ | 222 | ▐ | 254 | ■ |
| 031 | (cursor down) | US | 063 | ? | 095 | _ | 127 | ⌂ | 159 | ƒ | 191 | ┐ | 223 | ▀ | 255 (blank 'FF') | |

注：(ISO　646)，128～255 是 IBM-PC(长城 0520)上专用的。表中 000～127 是标准的。

# 附录Ⅱ MS BASIC语句和函数一览表

1. **命令一览表**

| 命　令 | 功　能 |
|---|---|
| AUTO | 自动产生行号 |
| BLOAD | 将二进制数据（例如机器语言程序）装入内存 |
| BSAVE | 将二进制数据送到磁盘保存 |
| CLEAR | 清除程序变量，释放内存区 |
| CONT | 继续运行程序 |
| DELETE | 删除指定的程序行 |
| EDIT | 显示出需要修改的程序行 |
| FILES | 列出文件目录 |
| KILL | 删除文件 |
| LIST | 显示程序清单 |
| LITST | 打印程序清单 |
| LOAD | 装入一个程序 |
| MERCE | 将保存在软盘上的程序和在内存中的程序拼接起来 |
| NAME…AS… | 将软盘文件改名 |
| NEW | 清除内存中的当前程序和变量 |
| RENUM | 重编程序行号 |
| RESET | 初始化软盘信息 |
| RUN | 运行程序 |
| SAVE | 将内存中当前程序保存在盘上 |
| SYSTEM | 结束 BASIC，关闭所有文件，返回 DOS |
| TRON | 置跟踪状态 |
| TROFF | 脱离跟踪状态 |

2. 语句一览表

A. 非输入/输出语句

| 语句 | 功能 |
| --- | --- |
| CALL | 调用机器语言程序 |
| CHAIN | 链接程序。调用一个程序并将变量传递给它 |
| COM (n) ON/OFF/STOP | 通讯功能的开启和停止 |
| COMMON | 给链接程序传递变量 |
| DATE $ | 置日期 |
| DEF FN | 定义数值或字符串函数 |
| DEF type | 定义变量类型，type 为 INT. SNG. DBL. STR 之一 |
| DEF SEC | 定义内存用户可用区间 |
| DEFUSR | 定义机器语言子程序的起始地址 |
| DIM | 说明数组可用的最大下标值并分配空间 |
| END | 结束程序 |
| ERASE | 清除数组空间 |
| ERROR | 显示某一错误代码的信息 |
| FOR | 设置并执行循环 |
| GOSUB | 转子程序 |
| GOTO | 无条件转移 |
| IF…THEN…ELSE | 选择控制 |
| KEY ON/OFF/LIST | 显示软按键（功能键）或关闭显示软按键 |
| KEY | 设置软按键 |
| KEY (n) ON/OFF/STOP | 启停功能键或光标控制键 |
| LET | 赋值 |
| MID $ | 将一个字符串中的一部分（子串）用另一个字符串来代替 |
| MOTOR | 启停磁带马达 |
| NEXT | 循环出口 |
| ON COM (n) GOSUB | 根据通信缓冲区的信息转子程序 |
| ON ERROR GOTO | 出错转移 |
| ON…GOSUB | 控制转移（执行子程序） |
| ON…GOTO | 控制转移 |
| ON KEY (n) GOSUB | 根据功能键或光标控制键转子程序 |
| ON PEN GOSUB | 根据光笔状态转子程序 |
| ON STRIG (n) GOSUB | 根据游戏操纵杆状态转子程序 |
| OPTION BASE | 定义数组下标下限 |
| PEN ON/OFF/STOP | 启停光笔功能 |
| POKE | 向内存指定地址写入数据 |
| RANDOMIZE | 重置启动随机数发生器 |
| REM | 注释 |
| RESTORE | 恢复数据区 |
| RESUME | 由出错子程序返回 |
| RETURN | 从子程序返回 |
| STOP | 程度暂停执行 |
| STRIG ON/OFF | 启停游戏操纵杆功能 |
| STRIG (n) ON/OFF/STOP | 启停游戏操纵杆 n 的功能 |
| SWAP | 两变量交换值 |
| TIME $ | 设置时间 |
| WAIT | 暂停执行等待端口信号 |
| WEND | WHILE 循环的出口 |
| WHILE | 当条件为真时执行循环 |

B. 输入/输出语句

| 语　　句 | 功　　能 |
|---|---|
| BEEP | 喇叭发声 |
| CIRCLE | 画圆 |
| CLOSE | 关闭文件 |
| CLS | 清屏 |
| COLOR | 设置屏幕颜色 |
| DATA | 置数据区数据 |
| DRAW | 绘图形 |
| FIELD | 设置随机文件缓冲区串变量及字节数 |
| GET# | 读随机文件记录 |
| GET | 从屏幕读绘图信息 |
| INPUT | 键盘输入 |
| INPUT# | 读文件中的数据 |
| LINE | 在屏幕上画直线 |
| LINE INPUT# | 从文件读一个完整行 |
| LINE INPUT | 从键盘读一个完整行 |
| LPRINT | 从宽行打印机输出 |
| LPRINT USING | 格式打印输出 |
| LSET | 给随机文件缓冲区串变量按左对齐赋值 |
| OPEN | 打开数据文件 |
| OPEN "COM…" | 打开通讯文件 |
| OUT | 送数据到端口 |
| PAINT | 屏幕着色 |
| PLAY | 放音乐 |
| PRINT | 屏幕显示 |
| PRINT USING | 格式显示 |
| PRINT# | 向文件写数据 |
| PRINT# USING | 按格式输出给文件 |
| PRESET | 在屏幕上用背景色画一个点 |
| PSET | 在屏幕上画一个点，如未指定颜色则用前景色画点 |
| PUT# | 向随机文件写数据 |
| PUT | 给屏幕送绘图信息 |
| READ | 从数据区读数 |
| RSET | 给随机文件缓冲区串变量按右对齐赋值 |
| SCREEN | 置屏幕显示类型 |
| SOUND | 喇叭发声 |
| WIDTH | 置屏幕宽 |
| WRITE | 在屏幕上输出 |
| WRITE# | 输出数据到文件 |

## 3. 函　数

### A. 算术函数

| 函　数 | 功　能 | 函　数 | 功　能 |
|---|---|---|---|
| ABS (X) | 取 x 的绝对值 | INT (X) | 不大于 x 的最大整数 |
| ATN (X) | 反正切 | LOG (X) | lnx |
| CDBL (X) | 转换成双精度 | RND (X) | 产生一个 (0, 1) 间的随机数 |
| CINT (X) | 转换 x 成整型 | SGN (X) | 取 x 的符号 |
| COS (X) | COSx | SIN (X) | sinx |
| CSNG (X) | 转换 x 成单精度 | SQR (X) | $\sqrt{x}$ |
| EXP (X) | $e^x$ | TAN (X) | tanx |
| FIX (X) | X 截尾取整 | | |

### B. 与串有关的数值函数

| 函　数 | 功　能 |
|---|---|
| ASC (X$) | 取 X$ 中第一个字符的 ASCII 码 |
| CVI (X$), CVS (X$), CVD (X$) | 将随机文件缓冲区代表数值的串变量变为整型、单精度型、双精度型数 |
| INSTR (n, X$, Y$) | 求出子串 X$ 在 Y$ 中的位置 |
| LEN (X$) | X$ 的长度 |
| VAL (X$) | 将 X$ 变成数值量 |

### C. 输入、输出和其它用途的函数

| 函　数 | 功　能 |
|---|---|
| CSRLIN | 得到光标的垂直位置 |
| EOF (f) | 指示文件 f 的文件结束状态 |
| IRL | 取最后产生错误的行号 |
| ERR | 取最后一次错误的错误代码 |
| FRE (X$) | 当前内存中自由空间 |
| INP (n) | 从端口 n 读一个字节 |
| LOC (f) | 末次读写记录的位置 |
| LOF (f) | 文件长度 (128 的整数倍) |
| LPOS (n) | 打印机的打印头位置 |
| PEEK (n) | 读内存地址 n 的一个字节 |
| PEN (n) | 读光笔 |
| POINT (X, Y) | 取点 (X, Y) 的颜色 |
| POS (n) | 光标列位置 |
| SCREEN (列、行) | 得到指定位置上的字符的 ASCII 码 |
| STICK (n) | 取游戏操纵杆坐标 |
| STRJG (n) | 取游戏操纵杆状态 |
| USR (X) | 调用机器语言子程序，自变量为 X |
| VARPTR (X) | 取变量 x 在内存中地址 |
| VARPTR (#f) | 取文件控制块地址 |

## D. 串函数（返回串值）

| 函 数 | 功 能 |
|---|---|
| CHP $ (n) | 求 ASCII 码为 n 的字符 |
| LEFT $ (X $ , n) | 取 X $ 左端 n 个字符 |
| MID $ (X $ , n, m) | 取 X $ 中第 n 字符开始的 m 个字符 |
| RIGHT $ (X $ , n) | 取 X $ 中右端 n 个字符 |
| SPACE $ (n) | 得 n 个空格的串 |
| STRING $ (n, m) | 取 ASCII 码为 m 的字符 n 次 |
| STRING $ (n, X $ ) | 取 X $ 的第一个字符 n 次 |

## E. 输入、输出和其它类型

| 函 数 | 功 能 |
|---|---|
| DATE $ | 取系统日期 |
| HEX $ (n) | 把 n 转换成十六进制的字符 |
| INKEY $ | 从键盘读一个字符 |
| INPUT $ (n, #f) | 从文件 f 读 n 个字符 |
| MKI $ (x), MKS $ (x), MKD $ (x) | 将整型、单精度型、双精度型数转换成随机文件缓冲区串变量 |
| OCT $ (n) | 将 n 转换成八进制字串 |
| SPC $ (n) | 打印 n 个空格 |
| XTR $ (x) | 将 x 转换成字串 |
| TAB (n) | 在第 n 位置上开始打印 |
| TIME $ | 取系统时间 |
| VARPTPS (v) | 取变量类型、地址 |

# 参 考 文 献


[1] 谭浩强、田淑清编著，《BASIC 语言（第三次修订本）》，科学普及出版社，1987 年。

[2] 谭浩强、傅金铎编著，《结构化程序设计及其在 COBOL 中的实现》，清华大学出版社，1989 年。

[3] 谭浩强、张基温编著，《BASIC 程序设计教程》，高等教育出版社，1988 年。

[4] 谭浩强、张基温编著，《True BASIC 程序设计》，清华大学出版社，1989 年。

[5] 施伯乐等《结构程序设计基础》，浙江科技出版社，1989 年。

[6] N. Wirth, "Algorithms+Data structures=Programs", Prentice-Hall, 1976

[7] Rina Yarmish and Joshua Yarmish, "Problem Solving with ANSI structured BASIC"

[8] John G. Kemeng and Thomas E. Kurtg, "True BASIC", Addisonwesley, 1985（中译本：白光野等译，《结构式语言 True BASIC》，科学普及出版社，1988 年）。

[9] 谭浩强，《要全面地、正确地看待 BASIC 语言》，计算机信息报，1989 年 10 月 10 日。

[10] B. W. Kernighan and P. J. Plauger, "The Elements of Programming Style" 中译本《程序设计技巧》，晏晓焰编译，清华大学出版社，1985 年。

[11] John G. Kemeny and Thomas E. Kurtz "Back to BASIC"，中译本《BASIC 的回顾—历史、讹误及未来》，刘瑞挺等译，谭浩强校，《计算机世界》印刷发行，1989 年。

[12] 《IBM—PC BASIC 程序设计语言》，路贵增等译，同济大学出版社，1985 年。

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):
{
  "filename": "QkFTSUPor63oqIDigJTigJTnu5PmnoTljJbnqIvluo/orr7orqHmlZnnqItfMTAxMTA0MDcuemlw",
  "filename_decoded": "BASIC\u8bed\u8a00\u2014\u2014\u7ed3\u6784\u5316\u7a0b\u5e8f\u8bbe\u8ba1\u6559\u7a0b_10110407.zip",
  "filesize": 21859317,
  "md5": "c9781df8aa340c35092a051ba16ea242",
  "header_md5": "9b56276821dd4281aaffcdaba698c356",
  "sha1": "7dc69aedc0668271fdf0002d35c4d8bf4b923305",
  "sha256": "289b28aaabfd0ce1b9d86e7554f7ab0b99278cce51bc35d5670c740ada37302a",
  "crc32": 2364207253,
  "zip_password": "",
  "uncompressed_size": 22439985,
  "pdg_dir_name": "BASIC\u2559\u2229\u2564\u2558\u00ed\u00ac\u00ed\u00ac\u255c\u00df\u2563\u2563\u2557\u00bb\u2502\u2560\u2568\u2265\u2554\u03a6\u255d\u255e\u255c\u2560\u2502\u2560_10110407",
  "pdg_main_pages_found": 277,
  "pdg_main_pages_max": 277,
  "total_pages": 289,
  "total_pixels": 2026896460,
  "pdf_generation_missing_pages": false
}